

The image is a large, symmetrical, abstract graphic composed of the letters 'S' and 'Y' arranged in a grid-like pattern. The letters are black on a white background. The overall shape is a large, stylized 'Y' or a complex letterform. The top part is a wide horizontal bar made of 'S's, with 'Y's forming the vertical stems. The bottom part is a wide horizontal bar made of 'S's, with 'Y's forming the vertical stems. The central part is a vertical column of 'Y's. The sides are composed of 'S's and 'Y's in a repeating pattern. The overall effect is a dense, textured, and highly structured composition.

10  
VO

```

LL          IIIIII          SSSSSSSS
LL          IIIIII          SSSSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SSSSSS
LL          II             SSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SSSSSS
LLLLLLLLLLLL IIIIIIII          SSSSSSSS
LLLLLLLLLLLL IIIIIIII          SSSSSSSS

```

(3)	239	CANCEL I/O ON CHANNEL
(4)	273	Handle Last Channel Deassign
(5)	332	FILL DIAGNOSTIC BUFFER
(6)	363	RELEASE I/O CHANNEL
(7)	418	REQUEST I/O CHANNEL
(8)	478	I/O Request Completion Processing for Class Drivers
(9)	526	I/O REQUEST COMPLETION PROCESSING
(10)	637	MOUNT VERIFICATION HELPER
(11)	670	INITIATE I/O FUNCTION ON DEVICE
(12)	708	Allocate Buffered Data Path
(14)	819	Release Buffered Data Path
(15)	904	REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
(16)	945	REQUEST UNIBUS MAP REGISTERS
(17)	980	ALLOCATE UNIBUS MAP REGISTERS
(18)	1087	Allocate a specific set of UNIBUS Map Registers
(19)	1196	Permanently Allocate UNIBUS Map Registers
(21)	1321	Release UNIBUS Map Registers
(23)	1529	RETURN TO CALLER
(24)	1548	WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
(25)	1582	WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
(26)	1618	ALLOCATE SYSTEM PAGE TABLE
(27)	1653	CONVERT DEVICE NAME AND UNIT
(28)	1934	BROADCAST TO A TERMINAL
(29)	2047	BROADCAST EMERGENCY MESSAGE TO CONSOLE
(30)	2131	SCAN THE I/O DATA BASE
(31)	2191	SCAN THE I/O DATA BASE BOTH PRIMARY & SECONDARY PATHS
(32)	2262	IOC\$CTRLINIT - Call driver controller init. routine
(33)	2327	IOC\$UNITINIT - Call driver unit init. routine
(34)	2406	Parse Device Name String
(35)	2587	Search I/O Database for Device
(36)	2751	Continue I/O Database Search
(37)	2801	Check UCB Against Search Rules
(38)	2901	IOC\$THREADCRB



```
0000 1 .TITLE IOSUBNPAG - NONPAGED I/O RELATED SUBROUTINES
0000 2 .IDENT 'V04-000'
0000 3 *****
0000 4 *
0000 5 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 6 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 7 * ALL RIGHTS RESERVED.
0000 8 *
0000 9 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 10 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 11 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 12 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 13 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 14 * TRANSFERRED.
0000 15 *
0000 16 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 17 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 18 * CORPORATION.
0000 19 *
0000 20 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 21 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 22 *
0000 23 *
0000 24 *****
0000 25
0000 26 D. N. CUTLER 13-JUN-76
0000 27
0000 28
0000 29 NONPAGED I/O RELATED SUBROUTINES
0000 30
0000 31 MODIFIED BY:
0000 32
0000 33 V03-038 WMC0004 Wayne Cardoza 23-Aug-1984
0000 34 Add routine for emergency message to console.
0000 35
0000 36 V03-037 WMC0003 Wayne Cardoza 14-Aug-1984
0000 37 Fix ROW0409 to restore the correct register.
0000 38
0000 39 V03-036 ACG0442 Andrew C. Goldstein, 7-Aug-1984 17:52
0000 40 Save RB in IOC$LAST_CHAN; fix order of tests in IOC$TESTUNIT
0000 41 for correct allocation and mount checks. Fix handling of
0000 42 lock value block on device lock in IOC$TESTUNIT.
0000 43
0000 44 V03-035 ROW0409 Ralph O. Weber 6-AUG-1984
0000 45 Fix release map registers processing of requests waiting for
0000 46 map registers. Restore saved fork registers -- including the
0000 47 PDT address -- before the calling IOC$ALOMAPUDA at
0000 48 REALLOC_CD_MAPREGS.
0000 49
0000 50 V03-034 TCM0006 Trudy C. Matthews 20-Jul-1984
0000 51 Add routine IOC$THREADCRB.
0000 52
0000 53 V03-033 WMC0002 Wayne Cardoza 03-May-1984
0000 54 Add support for MNTVERPND bit.
0000 55
0000 56 V03-032 RAS0300 Ron Schaefer 2-May-1984
0000 57 Change IOC$CVT_DEVNAM to only prefix cluster node names if
```

```
0000 58 : the DEV$V_NNM device characteristic is set in UCBSL_DEVCHAR2.
0000 59 : Add additional itemcode (4) to IOC$CVT_DEVNAM to provide
0000 60 : the device name string sans unit number.
0000 61 :
0000 62 :
0000 63 : V03-031 TMK0001 Todd M. Katz 23-Apr-1984
0000 64 : Remove the $LOGDEF data definitions.
0000 65 :
0000 66 : V03-030 RLRPDTADP Robert L. Rappaport 9-Apr-1984
0000 67 : Modify entypoints used for allocating and deallocating
0000 68 : Buffered Data Paths and UNIBUS Map Registers for UQPORTS (UDA),
0000 69 : to pickup pointer for ADP from PDT$$_ADP(R4).
0000 70 :
0000 71 : V03-029 ACG0414 Andrew C. Goldstein, 30-Mar-1984 15:49
0000 72 : Minor parse and searching fixes in IOC$SEARCH...
0000 73 : add IOC$V_ALLOC to force allocation
0000 74 :
0000 75 : V03-028 ACG0406 Andrew C. Goldstein, 16-Mar-1984 15:42
0000 76 : Fix bugs in searching for allocation class
0000 77 :
0000 78 : V03-027 ACG0399 Andrew C. Goldstein, 24-Feb-1984 22:28
0000 79 : Add IOC$LAST_CHAN subroutine, and move in internal I/O
0000 80 : database parse and search routines, so they can be called
0000 81 : by IPC.
0000 82 :
0000 83 : V03-026 RLRMAPSP Robert L. Rappaport 15-Feb-1984
0000 84 : Correct bug in BEQL destination in IOC$ALOUBAMAPSP that is
0000 85 : only triggered if the range specified, coincides with the
0000 86 : exact end of an extent of map registers.
0000 87 :
0000 88 : V03-025 ROW0292 Ralph O. Weber 4-FEB-1984
0000 89 : Fix branch displacements broken by movement of EXE$MOUNTVER to
0000 90 : SYSLOAxxx.
0000 91 :
0000 92 : V03-024 KPL0001 Peter Lieberwirth 7-Nov-1983
0000 93 : Add paths for new processors to CPUDISP invocation.
0000 94 :
0000 95 : V03-023 ROW0244 Ralph O. Weber 17-OCT-1983
0000 96 : Change the IOC$CVT_DEVNAM name string formation rules to
0000 97 : eliminate $1$TIA0: and other allocation class based names
0000 98 : for devices which can never be dual pathed. See routine
0000 99 : comments for details of current operation mode.
0000 100 :
0000 101 : V03-022 ROW0239 Ralph O. Weber 11-OCT-1983
0000 102 : Fix IOC$CVT_DEVNAM to not insert node name or trailing dollar
0000 103 : sign when node name is null. Also correct comments describing
0000 104 : the R4 argument to IOC$CVT_DEVNAM.
0000 105 :
0000 106 : V03-021 ROW0234 Ralph O. Weber 5-OCT-1983
0000 107 : Change IOC$CVT_DEVNAM to produce $allocation-class$device
0000 108 : strings completely in ASCII, when allocation class output is
0000 109 : requested. In the process rip up the whole thing because that
0000 110 : was the only way to get something that worked and didn't
0000 111 : occupy all non-page memory
0000 112 :
0000 113 : V03-020 TCM0005 Trudy C. Matthews 5-OCT-1983
0000 114 : Add IOC$SCAN_IODB_2P which is functionally the same as
0000 : IOC$SCAN_IOC8 except that both primary and secondary paths to
```



0000 115 : a device are scanned.  
0000 116 :  
0000 117 :  
0000 118 : V03-019 KDM0084 Kathleen D. Morse 26-Sep-1983  
0000 119 : Added MicroVAX I support to CPUDISP macros.  
0000 120 :  
0000 121 : V03-018 ROW0221 Ralph O. Weber 8-SEP-1983  
0000 122 : Change IOCSUNITINIT to look for a unit initialization routine  
0000 123 : in the DDT before looking in the CRB. See the note in the  
0000 124 : routine's header for details.  
0000 125 :  
0000 126 : V03-017 ROW0203 Ralph O. Weber 5-AUG-1983  
0000 127 : Add two new routines IOCSCTRLINIT and IOCSUNITINIT. These are  
0000 128 : the proscribed mechanism for calling device driver controller  
0000 129 : and unit initialization routines. These routines correctly  
0000 130 : setup for, locate, and call the appropriate driver routines.  
0000 131 :  
0000 132 : V03-016 TCM0004 Trudy C. Matthews 26-Jul-1983  
0000 133 : Change IOCSVT\_DEVNAM to return the <allocation\_class>+  
0000 134 : <devnam> form of device name if R4 > 0.  
0000 135 :  
0000 136 : V03-015 RLRBYTEOFF Robert L. Rappaport 27-Jun-1983  
0000 137 : Correct error in IOCSREQDATAPUDA. Error is that this  
0000 138 : routine has operated in a NOWAIT mode, that is, if no  
0000 139 : Buffered Datapath was available, we just used the  
0000 140 : Direct Datapath. Unfortunately, this doesn't work on  
0000 141 : 780's and 790's if the user buffer is located at an  
0000 142 : odd byte address since Byte Offset doesn't work on the  
0000 143 : Direct Datapath for the UNIBUS Adapters on these  
0000 144 : processors.  
0000 145 :  
0000 146 : V03-014 LMPBUILD L. Mark Pilant, 26-Jun-1983 23:11  
0000 147 : Change references from TTY\$K\_WB\_HDRLEN to TTY\$K\_WB\_LENGTH.  
0000 148 :  
0000 149 : V03-013 TCM0003 Trudy C. Matthews 17-Jun-1983  
0000 150 : Change the way cluster-style device names are conditionally  
0000 151 : returned, such that cluster-style names are returned for  
0000 152 : local disk devices if the system is participating in a  
0000 153 : cluster (routine IOCSVT\_DEVNAM).  
0000 154 :  
0000 155 : V03-012 TCM0002 Trudy C. Matthews 09-Jun-1983  
0000 156 : Fix bug in TCM0001.  
0000 157 :  
0000 158 : V03-011 TCM0001 Trudy C. Matthews 21-Apr-1983  
0000 159 : Add new parameter to IOCSVT\_DEVNAM that allows caller  
0000 160 : to specify whether he wants the node name returned for  
0000 161 : local devices or not.  
0000 162 :  
0000 163 : V03-010 ROW0188 Ralph O. Weber 30-APR-1983  
0000 164 : Fix broken branches to PMS\$ routines.  
0000 165 :  
0000 166 : V03-009 KTA3022 Kerbey T. Altmann 29-Dec-1982  
0000 167 : Enhance KTA3018. Add new routine to scan the IO  
0000 168 : data base and return the blocks.  
0000 169 :  
0000 170 : V03-008 ROW0140 Ralph O. Weber 18-NOV-1982  
0000 171 : Cause IOCS\$DALOCUBAMAP to give non-fatal INCONSTATE,  
"Inconsistent UBA data base" bugcheck if number of map

```
0000 172 : registers to deallocate is zero.
0000 173 :
0000 174 :
0000 175 : V03-007 MLJ0101 Martin L. Jack 11-Nov-1982
0000 176 : Add $SBDEF.
0000 177 :
0000 178 : V03-006 KTA3018 Kerbey T. Altmann 01-Nov-1982
0000 179 : Modify CVT_DEVNAME for new IO database.
0000 180 :
0000 181 : V03-005 ROW0130 Ralph O. Weber 5-OCT-1982
0000 182 : Remove IOC$DELMBOX whose functionality is replaced by new
0000 183 : routines in module UCBCREDEL.
0000 184 :
0000 185 : V03-004 KDM0002 Kathleen D. Morse 28-Jun-1982
0000 186 : Added $DCDEF.
0000 187 :
0000 188 : V03-003 RLR0003 Robert L. Rappaport 1-June-1982
0000 189 : Correct errors in UNIBUS map register allocation and
0000 190 : deallocation that occur when the number of active
0000 191 : descriptors is zero. Errors were in IOC$ALOUBAMAPSP
0000 192 : (allocation error), IOC$ALOUBAPRM (allocation error),
0000 193 : and IOC$DALOCUBAMAP (deallocation error). The error
0000 194 : in IOC$DALOCUBAMAP is corrected in a patch to V3.1.
0000 195 :
0000 196 : V03-002 RLR0002 Robert L. Rappaport 22-May-1982
0000 197 : Remove IOC$REQMAPREGN and all comments that reference it.
0000 198 :
0000 199 : V03-001 RLR0001 Robert L. Rappaport 22-May-1982
0000 200 : Correct error in UNIBUS map register allocation that
0000 201 : doubly allocated registers when the number of active
0000 202 : descriptors was zero.
0000 203 : This bug corrected in patch to V3.1.
```



```
0000 205 :  
0000 206 :  
0000 207 : MACRO LIBRARY CALLS  
0000 208 :  
0000 209 :  
0000 210 $ADPDEF      :DEFINE ADP OFFSETS  
0000 211 $CADEF      :DEFINE CONDITIONAL ASSEMBLY PARAMETERS  
0000 212 $CANDEF      :DEFINE CANCEL I/O REASON CODES  
0000 213 $CDRPDEF     :DEFINE CLASS DRIVER I/O REQUEST PACKET  
0000 214 $CRBDEF      :DEFINE CRB OFFSETS  
0000 215 $DCDEF      :DEFINE DEVICE CLASSES  
0000 216 $ddbDEF      :DEFINE DDB OFFSETS  
0000 217 $DDTDEF      :DEFINE DDT OFFSETS  
0000 218 $DEVDEF      :DEFINE DEVICE CHARACTERISTICS FLAGS  
0000 219 $DYNDDEF     :DEFINE DYNAMIC POOL BLOCK TYPES  
0000 220 $EMBDEF      :DEFINE EMB OFFSETS  
0000 221 $IDBDEF      :DEFINE IDB OFFSETS  
0000 222 $IOCDEF      :DEFINE IOC$SEARCHxxx FLAGS  
0000 223 $IPLDEF      :DEFINE INTERRUPT PRIORITY LEVELS  
0000 224 $IRPDEF      :DEFINE IRP OFFSETS  
0000 225 $JIBDEF      :DEFINE JIB OFFSETS  
0000 226 $LCKDEF      :DEFINE LOCK MANAGER SYMBOLS  
0000 227 $MSCPDEF     :DEFINE MSCP STRUCTURES  
0000 228 $PCBDEF      :DEFINE PCB OFFSETS  
0000 229 $PDTDEF      :Define PDT offsets  
0000 230 $PRDEF       :DEFINE PROCESSOR REGISTERS  
0000 231 $PRVDEF      :DEFINE PRIVILEGE BITS  
0000 232 $SBDEF       : Define system block offsets  
0000 233 $SSDEF       :DEFINE SYSTEM STATUS CODES  
0000 234 $TTYDEF      :DEFINE TERMINAL WRITE PACKET OFFSETS  
0000 235 $SUBMDDEF     :Define UNIBUS Map Descriptor structure  
0000 236 $UCBDEF      :DEFINE UCB OFFSETS  
0000 237 $VECDEF      :DEFINE CRB VECTOR OFFSETS
```



```
0000 239 .SBTTL CANCEL I/O ON CHANNEL
0000 240 :+
0000 241 : IOC$CANCELIO - CANCEL I/O ON CHANNEL
0000 242 :
0000 243 : THIS ROUTINE IS A DEVICE INDEPENDENT CANCEL I/O ROUTINE THAT CONDITIONALLY
0000 244 : MARKS THE UCB SUCH THAT THE CURRENT I/O REQUEST WILL BE CANCELED IF CONDITIONS
0000 245 : WARRANT SUCH A ACTION.
0000 246 :
0000 247 : INPUTS:
0000 248 :
0000 249 : R2 = NEGATIVE OF THE CHANNEL NUMBER.
0000 250 : R3 = CURRENT IO PACKET.
0000 251 : R4 = PCB ADDRESS.
0000 252 : R5 = UCB ADDRESS.
0000 253 :
0000 254 : OUTPUTS:
0000 255 :
0000 256 : IF THE DEVICE IS BUSY, THE REQUEST IS FOR THE CURRENT PROCESS, AND
0000 257 : THE I/O WAS ISSUED FROM THE DESIGNATED CHANNEL, THEN THE CANCEL I/O
0000 258 : BIT IS SET IN THE CORRESPONDING UCB.
0000 259 :
0000 260 : R2, R3, R4, AND R5 ARE PRESERVED ACROSS CALL.
0000 261 : -
0000 262 :
0000 263 .PSECT WIONONPAGED
11 64 A5 08 E1 0000 264 IOC$CANCELIO:: :CANCEL I/O ON CHANNEL
60 A4 0C A3 D1 0005 265 BBC #UCB$V_BSY,UCB$W_STS(R5),10$ :IF CLR, DEVICE NOT BUSY
28 A3 52 B1 000A 266 CMPL IRP$L_PID(R3),PCB$L_PID(R4) :PROCESS ID MATCH?
64 A5 08 A8 0012 267 BNEQ 10$ :IF NEQ NO
05 0016 268 CMPW R2,IRP$W_CHAN(R3) :CHANNEL NUMBER MATCH
270 BNEQ 10$ :IF NEQ NO
271 10$: B1SW #UCB$M_CANCEL,UCB$W_STS(R5) :SET CANCEL PENDING
RSB ;
```

```
0017 273 .SBTTL Handle Last Channel Deassign
0017 274
0017 275 :+
0017 276 : IOC$LAST_CHAN - Last Channel Deassign Specific
0017 277 : IOC$LAST_CHAN_AMBX - Last Assoc. MBX Channel Deassign Specific
0017 278
0017 279 Functional Description:
0017 280
0017 281 Common functions done on last channel deassignment are handled. The
0017 282 driver's cancel I/O routine is called with an appropriate reason code
0017 283 (CAN$C_DASSGN for regular deassign, or CAN$C_AMBXDGN for associated
0017 284 mailboxes). If after the cancel routine finished UCB$V_DELETEUCB is
0017 285 set, the UCB is credited and deleted.
0017 286
0017 287 Inputs:
0017 288
0017 289 R5 UCB address
0017 290 R2 Channel index (LAST_CHAN only)
0017 291
0017 292 Outputs:
0017 293
0017 294 R0 thru R3 destroyed.
0017 295 If appropriate, UCB is deallocated.
0017 296
0017 297 :-
0017 298
0017 299 .ENABLE LSB
0017 300
0017 301 IOC$LAST_CHAN_AMBX::
0017 302 PUSH R8 ; Save R8
0019 303 CLR R2 ; Clear unused cancel inputs.
001B 304 MOVZBL #CAN$C_AMBXDGN, R8 ; Set cancel reason code.
001E 305 BRB 10$
0020 306
0020 307 IOC$LAST_CHAN::
0020 308 PUSH R8 ; Save R8
0022 309 MOVL UCB$L_IRP(R5), R3 ; Get active packet address.
0026 310 MOVZBL #CAN$C_DASSGN, R8 ; Set cancel reason code.
0029 311
0029 312 10$: MOVL UCB$L_DDT(R5), R0 ; Get DDT address.
002E 313 SETIPL UCB$B_FIPL(R5) ; Raise to fork IPL.
0032 314 JSB @DDT$C_CANCEL(R0) ; Call driver's cancel I/O routine.
0035 315 SETIPL #IPL$ASTDEL ; Lower IPL.
0038 316 BBS #DEV$V_ALL, - ; Branch if still allocated
003D 317 UCB$L_DEVCHAR(R5), 30$
003D 318 BITL #DEV$M_TRM!DEV$M_MBX, - ; Is this a terminal, remote terminal
0045 319 UCB$L_DEVCHAR(R5) ; or mailbox?
0045 320 BEQL 20$ ; Branch if not.
0047 321 BBSC #DEV$V_OPR, - ; Else, clear OPR bit.
004C 322 UCB$L_DEVCHAR(R5), 20$ ; This is an implicit operator disable.
004E 323 20$: BBC #UCB$V_DELETEUCB, - ; Branch if UCB not to be deleted.
004E 324 UCB$L_STS(R5), 30$
0051 325 BSBW IOC$CREDIT_UCB ; Else credit UCB quotas,
0054 326 BSBW IOC$DELETE_UCB ; and delete the UCB.
0057 327 30$: POPL R8 ; Restore R8
005A 328 RSB
005B 329
```



IOSUBNPAG  
V04-000

- NONPAGED I/O RELATED SUBROUTINES<sup>G 3</sup>  
Handle Last Channel Deassign

005B 330 .DISABLE LSB

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 8  
(4)

10  
V0

```
0058 332 .SBTTL FILL DIAGNOSTIC BUFFER
0058 333
0058 334 :+ IOC$DIAGBUFILL - FILL DIAGNOSTIC BUFFER
0058 335
0058 336 THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION, BUT BEFORE RELEASING
0058 337 THE I/O CHANNEL, TO FILL THE FINAL DEVICE PARAMETERS INTO AN INTERNAL DIAG-
0058 338 NOSTIC BUFFER IF ONE IS SPECIFIED.
0058 339
0058 340 INPUTS:
0058 341
0058 342 R4 = ADDRESS OF DEVICE CSR REGISTER.
0058 343 R5 = DEVICE UNIT UCB ADDRESS.
0058 344
0058 345 OUTPUTS:
0058 346
0058 347 IF A DIAGNOSTIC BUFFER WAS SPECIFIED IN THE ORIGINAL REQUEST, THEN
0058 348 THE COMPLETION TIME, FINAL ERROR COUNTERS, AND DEVICE REGISTERS ARE
0058 349 FILLED INTO THE DIAGNOSTIC BUFFER.
0058 350 :-
0058 351
0058 352 IOC$DIAGBUFILL::
0058 353 :FILL DIAGNOSTIC BUFFER
1B 53 5B A5 DO 005B 353 MOVL UCB$L_IRP(R5),R3 :GET ADDRESS OF I/O PACKET
2A A3 07 E1 005F 354 BBC #IRP$DIAGBUF,IRP$W_STS(R3),10$ :IF CLR, NO DIAGNOSTIC BUFFER
50 4C B3 DO 0064 355 MOVL @IRP$L_DIAGBUF(R3),R0 :GET ADDRESS OF INTERNAL BUFFER DATA AREA
50 08 CO 0068 356 ADDL #8,R0 :POINT PAST START TIME
80 00000000 EF 7D 006B 357 MOVQ EXESGQ_SYSTIME,(R0)+ :INSERT COMPLETION TIME
80 0080 C5 3C 0072 358 MOVZWL UCB$B_ERTCNT(R5),(R0)+ :INSERT FINAL ERROR COUNTERS
52 0088 C5 DO 0077 359 MOVL UCB$L_DDT(R5),R2 :GET ADDRESS OF DDT
10 B2 16 007C 360 JSB @DDT$L_REGDUMP(R2) :CALL DEVICE SPECIFIC REGISTER DUMP ROUTINE
05 007F 361 10$: RSB :
```



```
0080 363 .SBTTL RELEASE I/O CHANNEL
0080 364
0080 365 :+ IOCSRELCHAN - RELEASE ALL I/O CHANNELS
0080 366 : IOCSRELSCHAN - RELEASE SECONDARY I/O CHANNEL
0080 367
0080 368 THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION TO RELEASE ALL
0080 369 CHANNELS THE I/O WAS BEING PERFORMED ON.
0080 370
0080 371 INPUTS:
0080 372
0080 373 R5 = UCB ADDRESS OF DEVICE UNIT.
0080 374
0080 375 OUTPUTS:
0080 376
0080 377 THE CHANNELS ARE RELEASED AND AN ATTEMPT IS MADE TO REMOVE THE NEXT
0080 378 WAITING DRIVER PROCESS FROM EACH CHANNEL QUEUE. IF A DRIVER PROCESS
0080 379 IS WAITING, THEN THE CHANNEL IS ASSIGNED TO THAT DRIVER PROCESS AND
0080 380 IT IS CALLED VIA A JSB TO ITS CHANNEL WAIT RETURN ADDRESS. WHEN THE
0080 381 CALLED DRIVER PROCESS RETURNS, A RETURN IS MADE TO THE DRIVER PROCESS
0080 382 THAT RELEASED THE CHANNEL. IF THERE IS NO DRIVER PROCESS WAITING FOR
0080 383 THE CHANNEL, THEN THE CHANNEL STATUS IS SET TO IDLE.
0080 384
0080 385 R3 AND R4 ARE PRESERVED ACROSS CALL.
0080 386
0080 387
0080 388 .ENABL LSB
0080 389 IOCSRELSCHAN::
0080 390 MOVL UCB$$_CRB(R5),R0 :RELEASE SECONDARY I/O CHANNEL
0080 391 MOVL CRB$$_LINK(R0),R0 :GET ADDRESS OF PRIMARY CRB
0080 392 BRB 20$ :GET ADDRESS OF SECONDARY CRB
0080 393 IOCSRELCHAN::
0080 394 MOVL UCB$$_CRB(R5),R0 :RELEASE I/O CHANNEL
0080 395 MOVL CRB$$_LINK(R0),R0 :GET ADDRESS OF PRIMARY CRB
0080 396 BEQL 10$ :GET ADDRESS OF SECONDARY CRB
0080 397 BSBB 20$ :IF EQL NONE
0080 398 10$: MOVL UCB$$_CRB(R5),R0 :RELEASE SECONDARY CHANNEL
0080 399 20$: BBC #CRB$$_BSY,CRB$$_MASK(R0),30$ :GET ADDRESS OF PRIMARY CRB
0080 400 MOVL CRB$$_INTD+VEC$$_IDB(R0),R1 :IF CLR, THEN CHANNEL NOT BUSY
0080 401 CMPL R5,IDB$$_OWNER(RT),R1 :GET ADDRESS OF IDB
0080 402 BNEQ 30$ :DRIVER PROCESS OWN CHANNEL?
0080 403 REMQUE @CRB$$_WQFL(R0),R2 :IF NEQ NO
0080 404 BVS 40$ :GET ADDRESS OF NEXT DRIVER FORK BLOCK
0080 405 PUSHF #M<R3,R4,R5> :IF VS NO DRIVER PROCESS WAITING
0080 406 MOVL R2,R5 :SAVE CONTEXT OF CURRENT DRIVER PROCESS
0080 407 MOVL UCB$$_FR3(R5),R3 :COPY ADDRESS OF DRIVER PROCESS FORK BLOCK
0080 408 MOVL IDB$$_CSR(R1),R4 :LOAD WAITING DRIVER PROCESS CONTEXT
0080 409 MOVL R5,IDB$$_OWNER(R1) :SET ASSIGNED CHANNEL CSR ADDRESS
0080 410 JSB @UCB$$_FPC(R5) :SET ADDRESS OF OWNER PROCESS UCB
0080 411 POPR #M<R3,R4,R5> :CALL DRIVER AT CHANNEL WAIT RETURN ADDRESS
0080 412 30$: RSB :RESTORE PREVIOUS DRIVER PROCESS CONTEXT
0080 413 40$: CLRL IDB$$_OWNER(R1) :CLEAR OWNER UNIT UCB ADDRESS
0080 414 BICB #CRB$$_BSY,CRB$$_MASK(R0) :CLEAR CHANNEL BUSY
0080 415 RSB
0080 416 .DSABL LSB
```

```
00CD 418 .SBTTL REQUEST I/O CHANNEL
00CD 419
00CD 420 :+ IOC$REQPCHANH - REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
00CD 421 : IOC$REQSCHANH - REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
00CD 422 : IOC$REQPCHANL - REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
00CD 423 : IOC$REQSCHANL - REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
00CD 424
00CD 425 : THESE ROUTINES ARE CALLED TO REQUEST AN I/O CHANNEL TO PERFORM AN I/O
00CD 426 : OPERATION ON.
00CD 427
00CD 428 : INPUTS:
00CD 429
00CD 430 : R5 = UCB ADDRESS OF DEVICE UNIT.
00CD 431 : 04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
00CD 432
00CD 433 : OUTPUTS:
00CD 434
00CD 435 : IF THE SPECIFIED I/O CHANNEL IS IDLE, THEN IT IS IMMEDIATELY
00CD 436 : ASSIGNED TO THE CURRENT DRIVER PROCESS. ELSE THE DRIVER PROCESS
00CD 437 : CONTEXT IS SAVED IN ITS FORK BLOCK, THE FORK BLOCK IS INSERTED
00CD 438 : IN THE CHANNEL WAIT QUEUE, AND A RETURN TO THE DRIVER PROCESS'
00CD 439 : CALLER IS EXECUTED.
00CD 440
00CD 441 : WHEN THE CHANNEL IS ASSIGNED, THE CSR ADDRESS OF THE ASSIGNED
00CD 442 : CONTROLLER IS RETURNED TO THE CALLER IN REGISTER R4.
00CD 443
00CD 444 : R3 IS PRESERVED ACROSS CALL.
00CD 445 :-
00CD 446
00CD 447 .ENABL LSB
00CD 448 IOC$REQSCHANH:: : REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
50 24 A5 DO 00CD 449 : GET ADDRESS OF PRIMARY CRB
50 20 A0 DO 00D1 450 : GET ADDRESS OF SECONDARY CRB
OE 11 00D5 451 :
00D7 452 IOC$REQSCHANL:: : REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
50 24 A5 DO 00D7 453 : GET ADDRESS OF PRIMARY CRB
50 20 A0 DO 00DB 454 : GET ADDRESS OF SECONDARY CRB
OD 11 00DF 455 :
00E1 456 IOC$REQPCHANH:: : REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
50 24 A5 DO 00E1 457 : GET ADDRESS OF PRIMARY CRB
52 50 DO 00E5 458 10$: : SET ADDRESS OF WAIT QUEUE LISTHEAD
08 08 11 00E8 459 :
00EA 460 IOC$REQPCHANL:: : REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
50 24 A5 DO 00EA 461 : GET ADDRESS OF PRIMARY CRB
52 04 A0 DO 00EE 462 20$: : GET ADDRESS OF LAST ENTRY IN QUEUE
51 2C A0 DO 00F2 463 30$: : GET ADDRESS OF IDB
08 0E A0 00 E2 00F6 464 : IF SET, THEN CHANNEL BUSY
54 61 DO 00FB 465 : SET ASSIGNED CHANNEL CSR ADDRESS
04 A1 55 DO 00FE 466 : SET OWNER UCB ADDRESS
05 0102 467 :
10 A5 53 DO 0103 468 40$: : SAVE R3 IN FORK BLOCK
OC A5 8ED0 0107 469 : SAVE CHANNEL WAIT RETURN ADDRESS
62 65 OE 010B 470 : INSERT DRIVER PROCESS IN CHANNEL WAIT
04 A1 55 D1 010E 471 : CURRENT DRIVER PROCESS OWNER?
03 12 0112 472 : IF NEQ, BRANCH TO RETURN
FF73 31 0114 473 : IF EQL BRW TO RELEASE CHANNELS
0117 474 50$: :

```



IOSUBNPAG  
V04-000

- NONPAGED I/O RELATED SUBROUTINES<sup>K 3</sup>  
REQUEST I/O CHANNEL

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 12  
(7)

05 0117 475 RSB  
0118 476 .DSABL LSB

;

```
0118 478 .SBTTL I/O Request Completion Processing for Class Drivers
0118 479
0118 480 :+
0118 481 : IOC$ALTREQCOM - I/O Request Complete Alternate Entry.
0118 482 :
0118 483 : This routine is entered when an I/O operation is completed on one
0118 484 : one of the devices using the disk or tape class drivers.
0118 485 : The packet is inserted in the I/O finish queue for I/O post
0118 486 : processing.
0118 487 :
0118 488 : INPUTS:
0118 489 :
0118 490 : R0 = First longword of I/O status
0118 491 : R1 = Second longword of I/O status
0118 492 : R5 = CDRP address
0118 493 :
0118 494 : OUTPUTS:
0118 495 :
0118 496 : The I/O packet is inserted in the I/O Post Processing Queue,
0118 497 : a Software interrupt is requested to initiate I/O Post
0118 498 : Processing.
0118 499 :-
0118 500
0118 501 IOC$ALTREQCOM::
53 A0 A5 9E 0118 502 MOVAB CDRP$L_10QFL(R5),R3 : R3 => IRP section of CDRP. This is
0118 503 : for compatibility with rest of QIO
0118 504 : logic.
55 1C A3 D0 0118 505 MOVL IRP$L_UCB(R3),R5 : R5 => UCB.
70 A5 D6 0120 506 INCL UCB$L_OPCNT(R5) : Increment operations completed
0123 507
0123 508 BLBC R0,20$ : LBC implies I/O error, so goto call
0126 509 : MOUNT VERIFICATION just in case.
0126 510 10$:
38 A3 50 7D 0126 511 MOVQ R0,IRP$L_MEDIA(R3) : Save final I/O status in IRP.
012A 512
012A 513 .IF DF CAS_MEASURE_IOT
012A 514
012A 515 JSB G^PMS$END_IO : Insert end of I/O transaction message
0130 516
0130 517 .ENDC
0130 518
00000000'FF 63 0E 0130 519 INSQUE (R3),AL^IOC$GL_PSBL : Insert packet in POST process queue
0137 520 SOFTINT #IPL$_IOPOST : Initiate SOFTWARE INTERRUPT
05 013A 521 RSB
013B 522 20$:
00000000'GF 16 013B 523 JSB G^EXE$MOUNTVER : If LBC, call MOUNT VERIFICATION.
E3 11 0141 524 BRB 10$ : Go back to normal flow.
```



```
0143 526 .SBTTL I/O REQUEST COMPLETION PROCESSING
0143 527
0143 528 :+ IOC$REQCOM - I/O REQUEST COMPLETE
0143 529
0143 530 : THIS ROUTINE IS ENTERED WHEN AN I/O OPERATION IS COMPLETED ON A
0143 531 : DEVICE UNIT. THE FINAL I/O STATUS IS STORED IN THE ASSOCIATED I/O
0143 532 : PACKET AND THE PACKET IS INSERTED IN THE I/O FINISH QUEUE FOR
0143 533 : I/O POST PROCESSING. DEVICE UNIT BUSY IS CLEARED AND AN ATTEMPT
0143 534 : IS MADE TO START ANOTHER I/O REQUEST ON THE DEVICE UNIT.
0143 535
0143 536 : IF THE I/O REQUEST COMPLETED WITH AN ERROR, AND THE DEVICE IS
0143 537 : A DISK, THEN BRANCH TO THE MOUNT VERIFICATION CODE, WHICH WILL
0143 538 : DETERMINE IF THE SITUATION REQUIRES MOUNT VERIFICATION.
0143 539
0143 540 : IF MOUNT VERIFICATION IS IN PROGRESS, NO FURTHER I/O REQUESTS WILL
0143 541 : BE INITIATED. THIS HAS A SIDE EFFECT OF KEEPING THE 'BSY' BIT IN
0143 542 : WHATEVER STATE IT IS CURRENTLY IN. FOR CONVENTIONAL DISK DRIVERS,
0143 543 : THE BSY BIT WILL BE LEFT ON, WHICH WILL BLOCK $QIO FROM INITIATING
0143 544 : ANY NEW I/O ON THE DEVICE. FOR THE DISK CLASS DRIVER, THE BUSY
0143 545 : BIT WILL BE OFF, WHICH WILL ALLOW $QIO TO INITIATE NEW I/O.
0143 546
0143 547 : INPUTS:
0143 548
0143 549 : R0 = FIRST LONGWORD OF I/O STATUS.
0143 550 : R1 = SECOND LONGWORD OF I/O STATUS.
0143 551 : R5 = UCB ADDRESS OF DEVICE UNIT.
0143 552
0143 553 : OUTPUTS:
0143 554
0143 555 : THE I/O PACKET IS INSERTED IN THE I/O POST PROCESSING QUEUE
0143 556 : AND DEVICE UNIT BUSY IS CLEARED. A SOFTWARE INTERRUPT IS
0143 557 : REQUESTED TO INITIATE I/O POST PROCESSING.
0143 558 :-
0143 559
0143 560 .ENABL LSB
0143 561 IOC$REQCOM:: :I/O DONE PROCESSING
0143 562 BBCC #UCB$V_ERLOGIP,UCB$W_STS(R5),10$ :IF CLR, ERROR LOG NOT IN PROGRESS
0148 563 MOVL UCB$L_EMB(R5),R2 :GET ADDRESS OF ERROR MESSAGE BUFFER
014D 564 MOVW UCB$W_STS(R5),EMB$W_DV_STS(R2) :INSERT FINAL DEVICE STATUS
0152 565 MOVW UCB$B_ERTCNT(R5),EMB$B_DV_ERTCNT(R2) :INSERT FINAL ERROR COUNTERS
0158 566 MOVQ R0,EMB$Q_DV_IOSB(R2) :INSERT FINAL I/O STATUS
015C 567 PUSHL R0 :SAVE R0
015E 568 BSBW ERL$RELEASEMB :RELEASE ERROR MESSAGE BUFFER
0161 569 POPL R0 :RESTORE R0
0164 570 10$: MOVL UCB$L_IRP(R5),R3 :GET ADDRESS OF I/O PACKET
0168 571 INCL UCB$L_OPCNT(R5) :INCREMENT OPERATIONS COMPLETED
016B 572 BLBC R0,DISKCHK :IF I/O ERROR, CHECK FOR DISK DEVICE
016E 573
016E 574 : DO NOT SAVE THE I/O STATUS IN THE IRP UNTIL IT HAS BEEN DECIDED THAT
016E 575 : MOUNT VERIFICATION IS NOT NECESSARY. THIS IS TO AVOID OVERWRITING THE
016E 576 : PHYSICAL DISK ADDRESS STORED IN THE IRP AT OFFSET IRP$L_MEDIA.
016E 577
016E 578 20$: MOVQ R0,IRP$L_MEDIA(R3) :STORE FINAL I/O STATUS
0172 579
0172 580 .IF DF CAS_MEASURE_IOT
0172 581
0172 582 TSTL L*PM$GL_IOPFMPDB :DATA COLLECTION ENABLED?
```

1C 64 A5 02 E5	0143 562
52 0094 C5 D0	0148 563
1A A2 64 A5 B0	014D 564
10 A2 0080 C5 B0	0152 565
12 A2 50 7D	0158 566
50 DD	015C 567
FE9F' 30	015E 568
50 BED0	0161 569
53 58 A5 D0	0164 570
70 A5 D6	0168 571
2A 50 E9	016B 572
	016E 573
	016E 574
	016E 575
	016E 576
	016E 577
38 A3 50 7D	016E 578
	0172 579
	0172 580
	0172 581
00000000'EF D5	0172 582

```
36 12 0178 583 BNEQ DO_PMS ;BRANCH IF YES
      017A 584
      017A 585 .ENDC
      017A 586
00000000'FF 63 0E 017A 587 PMSEND: INSQUE (R3),@L^IOCSGL_PSBL ;INSERT PACKET IN POST PROCESS QUEUE
      0181 588 SOFTINT #IPL$ IOPOST ;INITIATE SOFTWARE INTERRUPT
      0E E0 0184 589 BBS #UCBSV_MNTVERIP,- ;BRANCH IF MOUNT VERIFICATION IN PROGRESS
      2F 64 A5 0186 590 UCBSW_STS(R5),MNTVERPNDCHK ;(NOTE THIS LEAVES 'BSY' AS IS)
      53 4C B5 0F 0189 591 NXTIRP: REMQUE @UCBSL_IOQFL(R5),R3 ;REMOVE I/O PACKET FROM DEVICE UNIT QUEUE
      4C 1C 018D 592 BVC IOCS$INITIATE ;IF VC INITIATE NEXT FUNCTION
      64 A5 0100 8F AA 018F 593 BICW #UCBSM_BSY,UCBSW_STS(R5) ;CLEAR UNIT BUSY
      FEF2 31 0195 594 RELEASE: ;RELEASE ALL CHANNELS
      0195 595 BRW IOCS$RELCHAN ;
      0198 596
      0198 597 ; IF THIS IS A DISK DEVICE, CALL THE MOUNT VERIFICATION ROUTINE
      0198 598 ; TO DETERMINE IF MOUNT VERIFICATION IS NECESSARY. IF NOT, CONTROL
      0198 599 ; WILL RETURN, AND THE REQUEST WILL BE COMPLETED IN THE NORMAL MANNER.
      0198 600
      0198 601 DISKCHK:
      01 91 0198 602 CMPB #DC$ DISK,- ;IS THIS DEVICE A DISK?
      40 A5 019A 603 UCBSB_DEVCLASS(R5)
      D0 12 019C 604 BNEQ 20$ ;BRANCH IF NOT
      13 E5 019E 605 BBCC #UCBSV_MNTVERPND,- ;CHECK FOR MOUNT VERIFICATION PENDING
      05 64 A5 01A0 606 UCBSL_STS(R5),30$ ;IF NOT, JUST ENTER MOUNT VERIFICATION
      DE E5 01A3 607 BBCC #UCBSV_MNTVERIP,- ;CLEAR IN-PROGRESS BIT BEFORE CALL
      00 64 A5 01A5 608 UCBSL_STS(R5),30$ ;SO IT WILL REALLY START
      00000000'GF 16 01A8 609 30$: JSB G^EXE$MOUNTVER ;START MOUNT VERIFICATION
      BE 11 01AE 610 BRB 20$ ;COMPLETE I/O REQUEST
      01B0 611
      01B0 612 .IF DF CAS$_MEASURE_IOT
      01B0 613
      00000000'GF 16 01B0 614 DO_PMS: JSB G^PMS$END_IO ;INSERT END OF I/O TRANSACTION MESSAGE
      C2 11 01B6 615 BRB PMSEND ;REJOIN COMMON CODE
      01B8 616
      01B8 617 .ENDC
      01B8 618
      01B8 619 ; THE MOUNT-VERIFICATION-PENDING BIT IS USED TO INDICATE THAT A DISK SHOULD GO
      01B8 620 ; INTO MOUNT VERIFICATION AS SOON AS THE CURRENT I/O IS DONE. THIS IS INTENDED
      01B8 621 ; FOR USE IN A CLUSTER TO STALL I/O WHEN QUORUM IS LOST.
      01B8 622
      01B8 623 MNTVERPNDCHK:
      13 E5 01B8 624 BBCC #UCBSV_MNTVERPND,- ;CHECK FOR MOUNT VERIFICATION PENDING
      DB 64 A5 01BA 625 UCBSL_STS(R5),RELEASE ;IF NOT, JUST CLEAN UP
      01 91 01BD 626 CMPB #DC$ DISK,- ;IS THIS DEVICE A DISK?
      40 A5 01BF 627 UCBSB_DEVCLASS(R5)
      D2 12 01C1 628 BNEQ RELEASE ;BRANCH IF NOT
      DE E5 01C3 629 BBCC #UCBSV_MNTVERIP,- ;CLEAR IN-PROGRESS BIT BEFORE CALL
      00 64 A5 01C5 630 UCBSL_STS(R5),40$
      53 D4 01C8 631 40$: CLRL R3 ;NO IRP PASSED TO MOUNT VERIFICATION
      00000000'GF 16 01CA 632 JSB G^EXE$MOUNTVER ;TRY TO START MOUNT VERIFICATION
      B7 11 01D0 633 BRB NXTIRP ;WASN'T NECESSARY
      01D2 634
      01D2 635 .DSABL LSB
```

```
01D2 637 .SBTTL MOUNT VERIFICATION HELPER
01D2 638
01D2 639 :++
01D2 640 : IOC$MNTVER - Assist driver with mount verification.
01D2 641 : This routine is called by EXE$MOUNTVER to perform some driver-specific
01D2 642 : actions necessary for mount verification. This routine is used by non-
01D2 643 : CLASS drivers, and is called by default if EXE$MOUNTVER finds the address
01D2 644 : of IOC$RETURN in DDT$MNTVER.
01D2 645 :
01D2 646 : Inputs:
01D2 647 :
01D2 648 : R3 = IRP address or 0
01D2 649 : R5 = UCB address
01D2 650 :
01D2 651 : Outputs:
01D2 652 :
01D2 653 : None.
01D2 654 :
01D2 655 : Side effects:
01D2 656 :
01D2 657 : If R3 contains an IRP address, the IRP will be queued to the
01D2 658 : head of the UCB's IRP work queue. If R3 contains is zero, then
01D2 659 : remove the IRP from the head of the UCB's work queue and attempt
01D2 660 : to initiate the I/O.
01D2 661 :--
01D2 662 :
01D2 663 IOC$MNTVER::
01D2 664 TSTL R3 ;Driver-specific mount verification code
01D2 665 BEQL NXTIRP ;Check IRP address
01D2 666 INSQUE IRP$L_IOQFL(R3),- ;Branch if none
01D2 667 UCB$L_IOQFL(R5) ;Requeue the IRP
01D2 668 RSB ;Return
```

53 D5  
B3 13  
63 0E  
4C A5  
05 01DA



```
01DB 670 .SBTTL INITIATE I/O FUNCTION ON DEVICE
01DB 671 :+
01DB 672 : IOC$INITIATE - INITIATE NEXT FUNCTION ON DEVICE
01DB 673 :
01DB 674 : THIS ROUTINE IS CALLED TO INITIATE THE NEXT FUNCTION ON A DEVICE BY CLEARING
01DB 675 : STATUS BITS, SETTING THE OPERATION START TIME IF A DIAGNOSTIC BUFFER IS
01DB 676 : SPECIFIED, AND CALLING THE DRIVER AT ITS START I/O ENTRY POINT.
01DB 677 :
01DB 678 : INPUTS:
01DB 679 :
01DB 680 : R3 = ADDRESS OF I/O REQUEST PACKET.
01DB 681 : R5 = DEVICE UNIT UCB ADDRESS.
01DB 682 :
01DB 683 : OUTPUTS:
01DB 684 :
01DB 685 : CANCEL I/O, POWERFAIL, AND TIME OUT STATUS BITS ARE CLEARED, THE
01DB 686 : CURRENT SYSTEM TIME IS FILLED INTO THE INTERNAL DIAGNOSTIC BUFFER
01DB 687 : IF ONE IS SPECIFIED, AND THE DRIVER IS CALLED AT ITS START I/O ENTRY
01DB 688 : POINT.
01DB 689 : -
01DB 690 :
01DB 691 : IOC$INITIATE:: :INITIATE I/O FUNCTION
01DB 692 : MOVL R3,UCB$L_IRP(R5) :SAVE I/O PACKET ADDRESS
01DF 693 :
01DF 694 : .IF DF CAS_MEASURE_IOT
01DF 695 :
01DF 696 : JSB G^PMS$START_IO :INSERT START OF I/O TRANSACTION MESSAGE
01E5 697 :
01E5 698 : .ENDC
01E5 699 :
01E5 700 : MOVQ IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) :COPY TRANSFER PARAMETERS
01E5 701 : BICW #UCB$M_CANCEL!UCB$M_TIMEOUT,UCB$W_STS(R5) :CLEAR CANCEL AND TIME OUT
01E5 702 : BBC #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ :IF CLR, NO DIAGNOSTIC BUFFER
01E5 703 : MOVQ @IRP$L_DIAGBUF(R3),R0 :GET ADDRESS OF DIAGNOSTIC BUFFER DATA AREA
01E5 704 : MOVQ EXESGQ_SYSTIME,(R0) :INSERT I/O OPERATION START TIME
01E5 705 : MOVL UCB$L_DDT(R5),R0 :GET ADDRESS OF DRIVER DISPATCH TABLE
01E5 706 : JMP @DDT$C_START(R0) :START I/O OPERATION

58 A5 53 D0 01DB 691 IOC$INITIATE:: :INITIATE I/O FUNCTION
01DB 692 MOVL R3,UCB$L_IRP(R5) :SAVE I/O PACKET ADDRESS
01DF 693 :
01DF 694 .IF DF CAS_MEASURE_IOT
01DF 695 :
01DF 696 JSB G^PMS$START_IO :INSERT START OF I/O TRANSACTION MESSAGE
01E5 697 :
01E5 698 .ENDC
01E5 699 :
01E5 700 : MOVQ IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) :COPY TRANSFER PARAMETERS
01E5 701 : BICW #UCB$M_CANCEL!UCB$M_TIMEOUT,UCB$W_STS(R5) :CLEAR CANCEL AND TIME OUT
01E5 702 : BBC #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ :IF CLR, NO DIAGNOSTIC BUFFER
01E5 703 : MOVQ @IRP$L_DIAGBUF(R3),R0 :GET ADDRESS OF DIAGNOSTIC BUFFER DATA AREA
01E5 704 : MOVQ EXESGQ_SYSTIME,(R0) :INSERT I/O OPERATION START TIME
01E5 705 : MOVL UCB$L_DDT(R5),R0 :GET ADDRESS OF DRIVER DISPATCH TABLE
01E5 706 : JMP @DDT$C_START(R0) :START I/O OPERATION

78 A5 2C A3 7D 01E5 700 : MOVQ IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) :COPY TRANSFER PARAMETERS
64 A5 0048 8F AA 01E5 701 : BICW #UCB$M_CANCEL!UCB$M_TIMEOUT,UCB$W_STS(R5) :CLEAR CANCEL AND TIME OUT
0B 2A A3 07 E1 01E5 702 : BBC #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ :IF CLR, NO DIAGNOSTIC BUFFER
50 4C B3 D0 01E5 703 : MOVQ @IRP$L_DIAGBUF(R3),R0 :GET ADDRESS OF DIAGNOSTIC BUFFER DATA AREA
60 00000000'EF 7D 01E5 704 : MOVQ EXESGQ_SYSTIME,(R0) :INSERT I/O OPERATION START TIME
50 0088 C5 D0 01E5 705 : MOVL UCB$L_DDT(R5),R0 :GET ADDRESS OF DRIVER DISPATCH TABLE
00 B0 17 01E5 706 : JMP @DDT$C_START(R0) :START I/O OPERATION
```

```
0208 708 .SBTTL Allocate Buffered Data Path
0208 709
0208 710 :+ ALLOCATE BUFFERED DATA PATH CODE -
0208 711
0208 712 : IOCSREQDATAP - Entrypoint (called from traditional drivers) where caller
0208 713 : wishes to be queued (using UCB fork block) if no buffered data path
0208 714 : is available at the time of the call.
0208 715 : INPUT:
0208 716 : R5 => UCB.
0208 717
0208 718 : IOCSREQDATAPNW - Entrypoint to call when caller does not want to wait for
0208 719 : unavailable data path.
0208 720 : INPUT:
0208 721 : R5 => UCB
0208 722
0208 723 : IOCSREQDATAPUDA - Entrypoint (called from UDA port driver) where CDRP
0208 724 : is used as the source of information about the request and where
0208 725 : the caller does not want to wait for unavailable datapath.
0208 726
0208 727 : INPUT:
0208 728 : R4 => PDT
0208 729 : R5 => CDRP
0208 730 :-
0208 731
0208 732 : IOCSREQDATAP::
0208 733 : BSBB IOCSREQDATAPNW : Try to alloc. and get control after.
0208 734 : BLBS R0,10$ : LBS implies allocation success.
0208 735
0208 736 : MOVQ R3,UCBSL_FR3(R5) : Save driver context in UCB fork block.
0208 737 : POPL UCBSL_FPC(R5) : Save caller's return point.
0208 738 : INSQUE UCBSL_FQFL(R5) - : Queue fork block to resource wait queue.
0208 739 : @ADP$[_DPQBL(R1)] : Assumes IOCSALODATAP saves R1=>ADP.
0208 740 : RSB : Return to caller or caller's caller.
0208 741 : 10$:
0208 742 : IOCSREQDATAPNW::
0208 743 : MOVL UCBSL_CRB(R5),R0 : R0=>CRB.
0208 744 : MOVL CRBSL_INTD+VECSL_ADP(R0),R1 : R1=>ADP (pass to IOCSALODATAP)
0208 745 : MOVAB CRBSL_INTD+VECSW_MAPREG(R0),R2 : R2=>UBMD
0208 746
0208 747 : BRB IOCSALODATAP : NOWAIT, RSB from IOCSALODATAP
0208 748 : : returns to our caller.
0208 749 : IOCSREQDATAPUDA::
0208 750 : MOVL PDT$[_ADP(R4)],R1 : R1=>ADP (pass to IOCSALODATAP)
0208 751 : MOVAB CDRP$[_UBARSRCE(R5)],R2 : R2=>UBMD
0208 752
0208 753 : BSBB IOCSALODATAP : Call to allocate a data path.
0208 754 : BLBS R0,20$ : LBS means we got one.
0208 755 : BLBC CDRP$[_BOFF(R5)],20$ : LBC means, user buffer is on an
0208 756 : : even byte address so we can use
0208 757 : : the Direct Data Path.
0208 758
0208 759 : Here we have a transfer to a user buffer located at an odd byte address.
0208 760 : On those processors which support Byte Offset on the Direct Datapath, we
0208 761 : can continue processing. On other processors, we must wait for a buffered
0208 762 : datapath.
0208 763
0208 764 : CPUDISP <<780,10$>,- : On 11-780 we wait.
```

```
023A 765 <750,20$>,- ; On 11-750 we continue.
023A 766 <730,20$>,- ; On 11-730 we continue.
023A 767 <790,10$>,- ; On 11-790 we wait.
023A 768 <8SS,10$>,- ; On SCORPIO we wait.
023A 769 <8NN,10$>,- ; On NAUTILUS we wait.
023A 770 <UV1,30$>> ; On MicroVAX we bugcheck.
0254 771
10 A5 53 7D 0254 772 10$: MOVQ R3,CDRPSL FR3(R5) ; Save driver context in CDRP fork block.
OC A5 8ED0 0258 773 POPL CDRPSL_FPC(R5) ; Save caller's return point.
28 B5 B6 025C 774 INCW @CDRPSL_RWCPTR(R5) ; Increment RWAITCNT.
65 OE 025F 775 INSQUE CDRPSL_FQFL(R5),- ; Queue fork block to resource wait queue.
18 B1 0261 776 @ADPSL_DPQBL(R1) ; Assumes IOCSALODATAP saves R1=>ADP.
0263 777 20$:
05 0263 778 RSB ; Return to caller or caller's caller.
0264 779
0264 780 30$: BUG_CHECK IVBYTEALGN,FATAL
```



```
0268 782 : IOC$ALODATAP - Common subroutine called by above routines to allocate
0268 783 : a UNIBUS buffered datapath.
0268 784 :
0268 785 : INPUTS:
0268 786 : R1 => ADP wherein the datapath allocation bit map is stored.
0268 787 : R2 => UBA mapping descriptor in user's data structure.
0268 788 :
0268 789 : OUTPUTS:
0268 790 : R0 LBS - implies allocation success
0268 791 : datapath field in R2 => UBA mapping descriptor is set to the
0268 792 : number of the datapath allocated.
0268 793 : appropriate bit in datapath allocation bit map is cleared.
0268 794 : R0 LBC - implies allocation failure.
0268 795 :
0268 796 :
0268 797 : IOC$ALODATAP:
17 03 07 E0 0268 798 BBS #VECSV_PATHLOCK,- ; If this user has a permanently allocated
026A 799 UBMSB_DATAPATH(R2),10$ ; datapath, branch around to success.
026D 800
026D 801 ASSUME ADPSC_NUMDATAP EQ 16
026F 802 FFS #0,- ; Find first available datapath,
0270 803 #ADPSC_NUMDATAP,- ; according to bit map. Note failure
60 A1 0270 804 ADPSW_DPBITMAP(R1),- ; leaves R0 with the value '16', an
50 0272 805 R0 ; even number with the low bit clear.
12 13 0273 806 BEQL 20$ ; EQL implies failure.
0275 807
50 F0 0275 808 INSV R0,- ; Upon success, R0 has number of the
0277 809 #VECSV_DATAPATH,- ; available datapath to allocate.
05 00 0277 810 #VECSS_DATAPATH,- ; So we update the user's datapath
03 A2 0279 811 UBMSB_DATAPATH(R2) ; descriptor pointed at by R2.
027B 812
04 60 A1 50 E4 027B 813 BBSC R0,ADPSW_DPBITMAP(R1),10$; And we update the bit map.
0280 814 BUG_CHECK INCONSTATE ; We shouldn't be here obviously.
0284 815
50 01 D0 0284 816 10$: MOVL S^#SS$_NORMAL,R0 ; Indicate allocation success.
D5 0287 817 20$: RSB ; And we return to our caller.
```

```
0288 819 .SBTTL Release Buffered Data Path
0288 820
0288 821 :+
0288 822 : RELEASE BUFFERED DATA PATH CODE -
0288 823 : IOCSRELDATAPUDA - Entry point called from UDA port driver in response
0288 824 : to an UNMAP call. Here the data as to the buffered data path
0288 825 : is in the CDRP.
0288 826
0288 827 INPUTS:
0288 828 R4 => PDT
0288 829 R5 => CDRP
0288 830
0288 831 : IOCSRELDATAP - Entry point called from traditional drivers to release
0288 832 : the buffered datapath described in CRB$ _INTD+VEC$B_DATAPATH.
0288 833
0288 834 INPUTS:
0288 835 R5 => UCB
0288 836
0288 837 OUTPUTS:
0288 838 Datapath re-allocated (if any waiters). R0, R1, and R2 modified.
0288 839 NOTE: Since calls to IOCSREQDATAPUDA are NOWAIT, fork blocks dequeued
0288 840 here from ADP$ _DPQFL are guaranteed to be UCB's.
0288 841
0288 842 :
0288 843 : IOCSRELDATAPUDA::
0288 844 MOVL PDT$ _ADP(R4),R1 ; R1 => ADP.
0288 845 MOVAB CDRP$ _UBARSRC(R5),R2 ; R2 => UBMD.
0288 846 BRB RELDATAP_COMMON
0288 847 : IOCSRELDATAP::
0288 848 MOVL UCB$ _CRB(R5),R0 ; R0 => CRB.
0288 849 MOVAB CRB$ _INTD+VEC$ _MAPREG(R0),R2 ; R2 => UBMD.
0288 850 MOVL CRB$ _INTD+VEC$ _ADP(R0),R1 ; R1 => ADP.
0288 851 RELDATAP_COMMON:
0288 852 CVTBL UBMD$B_DATAPATH(R2),R0 ; Get datapath designator.
0288 853 BLEQ 10$ ; If LSS permanent assignment.
0288 854 ; If EQL we had NO datapath to
0288 855 ; release.
0288 856 INSV #0,- ; Zero datapath number.
0288 857 #VEC$V_DATAPATH,#VEC$S_DATAPATH,-
0288 858 UBMD$B_DATAPATH(R2)
0288 859 EXTZV #VEC$V_DATAPATH,- ; Extract datapath number.
0288 860 #VEC$S_DATAPATH,R0,R2
0288 861 REMQUE @ADP$ _DPQFL(R1),R0 ; R0 => next driver fork block
0288 862 BVS 20$ ; If VS no driver process waiting
0288 863
0288 864 MOVQ R3,-(SP) ; Save R3, R4, R5
0288 865 PUSHL R5
0288 866 MOVL R0,R5 ; R5 => driver fork block.
0288 867 CMPB #DYN$C_UCB,- ; See if we dequeued a UCB or a CDRP.
0288 868 UCB$B_TYPE(R5)
0288 869 BNEQ 30$ ; NEQ implies a CDRP.
0288 870
0288 871 ; Here we have R5 => UCB.
0288 872
0288 873 MOVL UCB$ _CRB(R5),R1 ; R1 => CRB.
0288 874
0288 875 INSV R2,- ; Store assigned datapath #
```

51	00E0	C4	D0	0288	844
52	3C	A5	9E	028D	845
		0C	11	0291	846
				0293	847
50	24	A5	D0	0293	848
52	34	A0	9E	0297	849
51	38	A0	D0	029B	850
				029F	851
50	03	A2	98	029F	852
		36	15	02A3	853
				02A5	854
				02A5	855
	05	00	F0	02A5	856
		00		02A7	857
	03	A2		02A9	858
		00	EF	02AB	859
52	50	05		02AD	860
50	14	B1	0F	02B0	861
		26	1D	02B4	862
				02B6	863
7E	53	7D		02B6	864
	55	DD		02B9	865
55	50	D0		02BB	866
	10	91		02BE	867
0A	A5			02C0	868
	22	12		02C2	869
				02C4	870
				02C4	871
				02C4	872
51	24	A5	D0	02C4	873
				02C8	874
	52	F0		02C8	875

```
05 00      02CA 876      #VECSV_DATAPATH,-      ; in CRB.
37 A1      02CA 877      #VECSS_DATAPATH,-
53 10 A5 7D 02CC 878      CRBSL_INTD+VECSB_DATAPATH(R1)
OC B5 16 02CE 879
53 55 8ED0 02CE 880      MOVQ UCBSL_FR3(R5),R3      ; Restore driver context.
OC B5 16 02D2 881      JSB @UCBSL_FPC(R5)      ; Call back waiting driver.
53 8E 7D 02D5 882 5$:      POPL R5      ; Restore deallocator's R5,R4,R3
FA 60 A1 52 05 02D8 883      MOVQ (SP)+,R3      ;
05 05 02DB 884      RSB      ; Return to deallocator.
E3 02DC 885 10$:      BBS R2,-      ;
05 02E1 886 20$:      ADPSW DPBITMAP(R1),10$      ; Set datapath bit and exit
05 02E1 887      BUG_CHECK INCONSTATE      ; Inconsistent state.
05 02E5 888      RSB      ;
05 02E6 889      ; Here we have R5 => CDRP.
05 02E6 890
05 02E6 891
05 02E6 892
05 02E6 893 30$:
52 F0 02E6 894      INSV R2,-      ; Store assigned datapath #
05 00      02E8 895      #VECSV_DATAPATH,-      ; in CDRP field.
3F A5      02E8 896      #VECSS_DATAPATH,-
00000000'EF 16 02EA 897      CDRPSL_UBARSRC+UBMDSB_DATAPATH(R5)
E1 11 02EC 898
02F2 900      JSB SCSSRESUMEWAITR      ; Resume waiting thread and any backed
02F2 901      BRB 5$      ; up IRP's.
02F4 902      ; Branch back to resume deallocator's
; thread.
```



```
02F4 904 .SBTTL REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
02F4 905 :+
02F4 906 IOC$REQMAPUDA - REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
02F4 907 :
02F4 908 THIS ROUTINE IS CALLED TO ALLOCATE UBA MAP REGISTERS AND TO MARK THE ALLOCATION
02F4 909 IN THE UBA MAP REGISTER ALLOCATION DATA STRUCTURES.
02F4 910 :
02F4 911 INPUTS:
02F4 912 :
02F4 913 R4 = ADDRESS OF PORT DESCRIPTOR TABLE.
02F4 914 R5 = ADDRESS OF CLASS DRIVER REQUEST PACKET (CDRP).
02F4 915 :
02F4 916 OUTPUTS:
02F4 917 :
02F4 918 IF MAP REGISTERS ARE ALLOCATED FOR THE CDRP, THE APPROPRIATE FIELDS
02F4 919 IN THE CDRP ARE MODIFIED TO INDICATE WHICH REGISTERS, AND THE NUMBER
02F4 920 OF REGISTERS THAT HAVE BEEN ALLOCATED. ALSO THE ALLOCATION DATA
02F4 921 STRUCTURE IN THE ADP IS MODIFIED.
02F4 922 :
02F4 923 IF MAP REGISTERS CANNOT BE ALLOCATED AT THIS TIME, THE CDRP IS
02F4 924 QUEUED ONTO THE RESOURCE WAIT LIST AND THE UCBSW_RWAITCNT IS
02F4 925 INCREMENTED.
02F4 926 :
02F4 927 :-
02F4 928 :
25 10 02F4 929 IOC$REQMAPUDA:: ; Allocate UBA map registers for class drive
02F4 930 BSBB IOC$ALOMAPUDA ; Call to allocate map registers if availabl
02F6 931 ; Returns R2 => ADP.
02F6 932 :
02F6 933 ; If here, low bit of R0 tells us whether we were successful in the allocation
02F6 934 attempt.
02F6 935 :
10 0F 50 E8 02F6 936 BLBS R0,10$ ; Branch around if successful.
10 A5 53 7D 02F9 937 MOVQ R3,CDRP$L_FR3(R5) ; Save driver process context
28 B5 B6 02FD 938 INCW @CDRP$L_RWCPTTR(R5) ; One more CDRP, on this UCB, awaiting
0C A5 8ED0 0300 939 ; resources.
65 OE 0304 940 POPL CDRP$L_FPC(R5) ; Save map register wait return address
34 B2 05 0306 941 INSQUE CDRP$L_FQFL(R5),-
0308 942 @ADP$L_MRQBL(R2) ; Insert process in map register wait queue
10$ 943 RSB ;
```

```
0309 945 .SBTTL REQUEST UNIBUS MAP REGISTERS
0309 946 :+
0309 947 IOCSREQMAPREG - REQUEST UNIBUS MAP REGISTERS
0309 948
0309 949 THIS ROUTINE IS CALLED TO REQUEST UNIBUS MAP REGISTERS TO PERFORM AN
0309 950 I/O TRANSFER.
0309 951
0309 952 INPUTS:
0309 953
0309 954 R5 = UCB ADDRESS OF DEVICE UNIT.
0309 955 04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
0309 956
0309 957 IT IS ASSUMED THAT THE CALLER OWNS THE I/O CHANNEL ON WHICH THE
0309 958 TRANSFER IS TO OCCUR ON.
0309 959
0309 960 OUTPUTS:
0309 961
0309 962 IF MAP REGISTERS HAVE BEEN PERMANENTLY ASSIGNED TO THE ASSOCIATED
0309 963 I/O CHANNEL, THEN CONTROL IS IMMEDIATELY RETURNED TO THE CALLER.
0309 964 ELSE AN ATTEMPT IS MADE TO ALLOCATE THE REQUESTED NUMBER OF MAP REG-
0309 965 ISTERS. IF SUFFICIENT CONTIGUOUS MAP REGISTERS ARE FOUND, THEN THEY
0309 966 ARE ASSIGNED TO THE ASSOCIATED I/O CHANNEL AND CONTROL IS RETURNED
0309 967 TO THE CALLER. ELSE THE DRIVER PROCESS CONTEXT IS SAVED IN ITS FORK
0309 968 BLOCK, THE FORK BLOCK IS INSERTED IN THE MAP REGISTER WAIT QUEUE,
0309 969 AND A RETURN TO THE DRIVER PROCESS' CALLER IS EXECUTED.
0309 970 :-
0309 971
0309 972 IOCSREQMAPREG::
0309 973 B5BB IOCSALOUBAMAP ;REQUEST UNIBUS MAP REGISTERS
0309 974 BLBS R0,10$ ; ALLOCATE UBA MAP REGISTER
0309 975 MOVQ R3,UCB$L_FR3(R5) ;IF LBS SUCCESSFUL ALLOCATION
0309 976 POPL UCB$L_FPC(R5) ;SAVE DRIVER PROCESS CONTEXT
0309 977 INSQUE UCB$L_FQFL(R5),@ADP$L_MRQBL(R2) ;SAVE MAP REGISTER WAIT RETURN ADDRESS
031A 978 10$: RSB ;INSERT PROCESS IN MAP REGISTER WAIT
                                ;
```

10	OC	3A	10
		50	E8
10	A5	53	7D
	OC	A5	8ED0
34	B2	65	0E
			05

```
031B 980 .SBTTL ALLOCATE UNIBUS MAP REGISTERS
031B 981
031B 982 :+
031B 983 : IOCSALOUBAMAP - ALLOCATE UBA MAP REGISTERS (CRB DATABASE SPECIFIED)
031B 984 : IOCSALOUBAMAPN - ALLOCATE UBA MAP REGISTERS (ARGUMENT SPECIFIED)
031B 985 : IOCSALOMAPUDA - ALLOCATE UBA MAP REGISTERS (FOR CLASS DRIVER(S))
031B 986
031B 987 This routine is called to allocate uba map registers and to mark the allocation
031B 988 in the map register allocation structure located in the ADP. The state
031B 989 of the UNIBUS map registers is maintained in a set of descriptors
031B 990 that describe contiguous extents of allocatable (i.e. free) map
031B 991 registers. A map register descriptor consists of the
031B 992 corresponding elements of two distinct arrays (of one word items)
031B 993 located in the ADP. These arrays, ADPSW_MRNREGARY and ADPSW_MRFREGARY,
031B 994 contain the number of map registers and the first map register in each
031B 995 contiguous extent of free map registers. These arrays are each
031B 996 preceded by a one word field containing all 1's (-1) so that compares
031B 997 made against the "previous" descriptor fail when the current descriptor
031B 998 is the one whose index is zero.
031B 999
031B 1000 ADPSL_MRACTMDRS maintains the number of active descriptors, i.e. the
031B 1001 number of elements of each array which contain valid data.
031B 1002
031B 1003 INPUTS: (FOR IOCSALOUBAMAP AND ALOUBAMAPN)
031B 1004 R3 = NUMBER OF MAP REGISTERS TO ALLOCATE (IOCSALOUBAMAPN only).
031B 1005 R5 = DEVICE UNIT UCB ADDRESS.
031B 1006
031B 1007 INPUT: (FOR IOCSALOMAPUDA)
031B 1008 R4 => PDT
031B 1009 R5 => CDRP
031B 1010
031B 1011 OUTPUTS:
031B 1012 R0 = SUCCESS INDICATION.
031B 1013 R2 => ADP
031B 1014
031B 1015 .enabl lsb
031B 1016 IOCSALOMAPUDA:
031B 1017 MOVQ R3,-(SP) ; Save R3,R4,R5
031B 1018 PUSHL R5 ;
031B 1019
031B 1020 MOVL PDT&L_ADP(R4),R2 ; R2 => ADP before we modify R4.
031B 1021
031B 1022 MOVL CDRPSL_BCNT(R5),R3 ; Get transfer byte count
031B 1023 MOVZWL CDRPSW_BOFF(R5),R4 ; Get byte offset in page
031B 1024 MOVAB ^X3FF(R3)[R4],R3 ; Calculate highest relative byte and round
031B 1025 ASHL #-9,R3,R3 ; Calculate number of map registers required
031B 1026
031B 1027 MOVAB CDRPSL_UBARSRC(R5),R1 ; R1 => UBMD.
031B 1028 BRB COMMON_ALOUBAMAP ; Branch to common code.
031B 1029
031B 1030 IOCSALOUBAMAPN:
031B 1031 MOVQ R3,-(SP) ; ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFIED
031B 1032 PUSHL R5 ; Save R3,R4,R5
031B 1033 BRB 5$ ;
031B 1034
031B 1035 IOCSALOUBAMAP:
031B 1036 MOVQ R3,-(SP) ; ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
031B 1037 PUSHL R5 ; Save R3,R4,R5
031B 1038
```

7E 53 7D 031B 1016  
55 DD 031E 1017  
52 00E0 C4 D0 0320 1018  
53 D2 A5 D0 0325 1020  
54 D0 A5 3C 0329 1021  
53 03FF C344 9E 032D 1023  
53 53 F7 8F 78 0333 1024  
51 3C A5 9E 0338 1025  
2F 11 033C 1027  
7E 53 7D 033E 1029  
55 DD 0341 1031  
18 11 0343 1032  
7E 53 7D 0345 1034  
55 DD 0348 1036



```

53 7E A5 3C 034A 1037
54 7C A5 3C 034A 1038
53 03FF C344 9E 0352 1039
53 53 F7 8F 78 0358 1040
51 24 A5 D0 035D 1041
52 38 A1 D0 035D 1042
51 34 A1 9E 0361 1043
38 0F E0 0365 1044
38 61 E0 0369 1045
036B 1046
036D 1047
036D 1048
036D 1049
036D 1050
036D 1051
036D 1052
036D 1053
036D 1054
036D 1055
5C A2 D5 036D 1056
13 13 0370 1057
53 53 D6 0372 1058
53 01 8A 0374 1059
55 55 D4 0377 1060
64 A245 53 B1 0379 1061
09 15 0379 1062
F4 55 5C A2 F2 037E 1063
50 D4 0380 1064
1F 11 0380 1065
0385 1066
0385 1067
0387 1068
0389 1069
61 015E C245 B0 0389 1070
038F 1071
02 A1 53 90 038F 1072
64 A245 53 A2 0393 1073
05 12 0398 1074
039A 1075
0129 30 039A 1076
06 11 039D 1077
039F 1078
015E C245 53 A0 039F 1079
03A5 1080
50 01 D0 03A5 1081
55 8E D0 03A8 1082
53 8E 7D 03AB 1083
05 05 03AE 1084
03AF 1085

MOVZWL UCBSW_BCNT(R5),R3 ;GET TRANSFER BYTE COUNT
MOVZWL UCBSW_BOFF(R5),R4 ;GET BYTE OFFSET IN PAGE
MOVAB ^X3FF(R3)[R4],R3 ;CALCULATE HIGHEST RELATIVE BYTE AND ROUND
ASHL #-9,R3,R3 ;CALCULATE NUMBER OF MAP REGISTERS REQUIRED

5$:
MOVL UCBSL_CRB(R5),R1 ; R1 => CRB.
MOVL CRBSL_INTD+VECSL_ADP(R1),R2 ; R2 => ADP.
MOVAB CRBSL_INTD+VECSW_MAPREG(R1),R1 ; R1 => UBMD.
BBS #VECSW_MAPLOCK,- ; If SET, already permanently
UBMD$W_MAPREG(R1),40$ ; allocated, so branch around.

; Here:
R1 => UBMD - caller's structure where we record registers allocated
R2 => ADP
R3 = number of map registers to allocate

COMMON_ALOUBAMAP:
TSTL ADPSL_MRACTMDRS(R2) ; Test for zero active descriptors.
BEQL 15$ ; EQL implies no registers available.
INCL R3 ; Round up request to next multiple
BICB #1,R3 ; of 2.
CLRL R5 ; Establish loop variable.

10$:
CMPW R3,ADPSW_MRNREGARY(R2)[R5] ; See if enough regs described here.
BLEQ 20$ ; LEQ implies YES.

AOBLSS ADPSL_MRACTMDRS(R2),R5,10$ ; Else branch back and continue

15$:
CLRL R0 ; If here, allocation failure.
BRB 50$ ; Branch around to return.

20$:
MOVW ADPSW_MRFREGARY(R2)[R5],- ; Allocate from low end of extent
UBMD$W_MAPREG(R1) ; by copying 1st map reg. #.
MOVB R3,UBMD$B_NUMREG(R1) ; Set # of map regs allocated.
SUBW R3,ADPSW_MRNREGARY(R2)[R5] ; Subtract out # regs allocated.
BNEQ 30$ ; NEQ implies extent not empty,
; branch around deallocate.
BSBW DEALLOC_DESCRIP ; Call to deallocate descriptor.
BRB 40$ ; And branch back to return.

30$:
ADDW R3,ADPSW_MRFREGARY(R2)[R5] ; Bump descriptor past
; allocated registers.

40$:
MOVL S^#SS$_NORMAL,R0 ; Indicate success.
50$:
POPL R5 ; Restore R5,R4,R3
MOVQ (SP)+,R3
RSB
.dsabl lsb
```

```
03AF 1087 .SBTTL Allocate a specific set of UNIBUS Map Registers
03AF 1088
03AF 1089 :+ IOCSALOUBAMAPSP
03AF 1090
03AF 1091 This routine is called to allocate a specific set of UNIBUS Map Registers.
03AF 1092
03AF 1093 INPUTS:
03AF 1094 R3 = # of map registers to allocate
03AF 1095 R4 = # of first map register to allocate
03AF 1096 R5 => UCB
03AF 1097
03AF 1098 OUTPUTS:
03AF 1099 R0 = Success or failure indication
03AF 1100 Note R0, R1 and R2 modified.
03AF 1101 :-
03AF 1102
03AF 1103 IOCSALOUBAMAPSP::
03AF 1104 MOVQ R3,-(SP) ; Save R3,R4,R5
03AF 1105 PUSHL R5 ;
03AF 1106
03AF 1107 MOVL UCBSL_CRB(R5),R0 ; R0 => CRB.
03AF 1108 MOVL CRBSL_INTD+VECSL_ADP(R0),R2 ; R2 => ADP.
03AF 1109 MOVAB CRBSL_INTD+VECSW_MAPREG(R0),R1 ; R1 => UBA mapping descriptor.
03AF 1110
03AF 1111 TSTL ADPSL_MRACTMDRS(R2) ; Test for zero active descriptors.
03AF 1112 BEQL 30$ ; EQL implies no registers available.
03AF 1113 BLBC R4,10$ ; Prepare to round DOWN to even boundary.
03AF 1114 BICB #1,R4 ; Clear low bit if set and
03AF 1115 INCL R3 ; then increment # of registers to allocate
03AF 1116 10$:
03AF 1117 INCL R3 ; Prepare to round UP to even # of registers
03AF 1118 BICB #1,R3 ;
03AF 1119
03AF 1120 CLRL R5 ; R5 will be index register.
03AF 1121 20$:
03AF 1122 CMPW R4,ADPSW_MRFREGARY(R2)[R5] ; Are registers we want in
03AF 1123 ; current extent?
03AF 1124 BLSS 30$ ; LSS means current extent is beyond the
03AF 1125 ; desired registers. Therefore they are
03AF 1126 ; not available and we have failed.
03AF 1127 BEQL 50$ ; EQL means they are at the beginning
03AF 1128 ; of the current extent.
03AF 1129
03AF 1130 ; Here the registers we want are either within the middle of the current
03AF 1131 ; extent or else beyond the current extent.
03AF 1132
03AF 1133 ADDW3 ADPSW_MRFREGARY(R2)[R5],- ; R0 = 1st register beyond
03AF 1134 ADPSW_MRNREGARY(R2)[R5],R0 ; current extent.
03AF 1135 CMPW R4,R0 ; Are we in current extent?
03AF 1136 BLSS 40$ ; LSS means YES, in current.
03AF 1137 AOBLS ADPSL_MRACTMDRS(R2),R5,20$ ; Loop thru all extents.
03AF 1138 30$: ; Failure if we fall thru.
03AF 1139 CLRL R0 ; Set failure code.
03AF 1140 BRB 80$ ; And branch to return.
03AF 1141 40$:
03AF 1142
03AF 1143 ; Here the first register we want is greater than the first register of
```

```
03F5 1144 : current extent (defined by R5 = index) and is less than or equal to
03F5 1145 : the last register of the extent. R0 contains the # of the register just
03F5 1146 : beyond the current extent. In other words,
03F5 1147 :
03F5 1148 : ADPSW_MRFREGARY(R2)[R5] < R4 < R0
03F5 1149 :
50 54 A2 03F5 1150 SUBW R4,R0 ; R0 = length of subextent based at R4.
53 50 B1 03F8 1151 CMPW R0,R3 ; Compare to # of registers needed.
F4 19 03FB 1152 BLSS 30$ ; LSS means failure.
03FD 1153
61 54 B0 03FD 1154 MOVW R4,UBMDSW_MAPREG(R1) ; Success. Fill in user's descriptor
02 A1 53 90 0400 1155 MOVW R3,UBMDSB_NUMREG(R1) ; with base register and # of registers.
0404 1156
0404 1157 : SUBW3 ADPSW_MRFREGARY(R2)[R5],R4,- ; Distance from beginning of
0404 1158 : ADPSW_MRNREGARY(R2)[R5] ; extent to R4 is new length.
64 A245 50 A2 0404 1159 SUBW R0,ADPSW_MRNREGARY(R2)[R5] ; Equivalent result.
0409 1160
50 53 A2 0409 1161 SUBW R3,R0 ; R0 = # regs. left in sub-extent.
36 13 040C 1162 BEQL 70$ ; EQL means we do not have to allocate
040E 1163 ; and fill a new extent descriptor.
55 D6 040E 1164 INCL R5 ; R5 = index of new extent descriptor.
7E 50 B0 0410 1165 MOVW R0,-(SP) ; Save length of new extent.
00C9 30 0413 1166 BSBW ALLOC_DESCRIP ; Call to allocate a new descriptor.
0416 1167
015E C245 53 54 A1 0416 1168 ADDW3 R4,R3,ADPSW_MRFREGARY(R2)[R5] ; Fill in new descriptor with
64 A245 8E B0 041D 1169 MOVW (SP)+,ADPSW_MRNREGARY(R2)[R5] ; 1st register and # registers.
20 11 0422 1170 BRB 70$ ; Branch around to success.
0424 1171 50$:
0424 1172 : Here the first register we want is equal to the first register of the current
0424 1173 : extent (defined by index register R5). In other words,
0424 1174 :
0424 1175 : R4 = ADPSW_MRFREGARY(R2)[R5]
0424 1176 :
64 A245 53 B1 0424 1177 CMPW R3,ADPSW_MRNREGARY(R2)[R5] ; See if we have enough registers.
C6 14 0429 1178 BGTR 30$ ; GTR implies failure.
042B 1180
61 54 B0 042B 1181 MOVW R4,UBMDSW_MAPREG(R1) ; Success. Fill in user's descriptor
02 A1 53 B0 042E 1182 MOVW R3,UBMDSB_NUMREG(R1) ; with 1st register and # allocated.
0432 1183
64 A245 53 A2 0432 1184 SUBW R3,ADPSW_MRNREGARY(R2)[R5] ; Update current descriptor.
08 13 0437 1185 BEQL 60$ ; EQL means current extent now
0439 1186 ; empty. Go to deallocate.
015E C245 53 A0 0439 1187 ADDW R3,ADPSW_MRFREGARY(R2)[R5] ; If not empty, update 1st register.
03 11 043F 1188 BRB 70$ ; Branch around deallocate.
0441 1189 60$:
0082 30 0441 1190 BSBW DEALLOC_DESCRIP ; Deallocate system descriptor.
50 01 D0 0444 1191 70$: MOVL S*#SS$_NORMAL,R0 ; Set success indicator.
55 8ED0 0447 1192 80$: POPL R5 ; Restore R5,R4,R3
53 8E 7D 044A 1193 MOVW (SP)+,R3 ;
05 044D 1194 RSB ; And return to caller.
```



```
044E 1196 .SBTTL Permanently Allocate UNIBUS Map Registers
044E 1197 :+
044E 1198 :IOC$ALOUBAMAPRM - Permanently Allocate UBA Map Registers (CRB Database Specified)
044E 1199 :IOC$ALOUBAMAPRMN - Permanently Allocate UBA Map Registers (Argument Specified)
044E 1200
044E 1201 This routine is called to permanently allocate UNIBUS map registers.
044E 1202 Here we allocate the map registers from the highest numbered
044E 1203 available registers.
044E 1204
044E 1205 INPUTS:
044E 1206 R3 = # Registers to allocate (IOC$ALOUBAMAPRMN only)
044E 1207 R5 => UCB
044E 1208
044E 1209 OUTPUTS:
044E 1210 R0 = Success indication
044E 1211 :-
044E 1212
044E 1213
044E 1214 .enabl LSB
044E 1215 IOC$ALOUBAMAPRMN::: :ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFIED
044E 1216 MOVQ R3,-(SP) : Save R3,R4,R5
044E 1217 PUSH R5 :
044E 1218
044E 1219 BRB 5$ :
044E 1220 IOC$ALOUBAMAPRM::: :ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
044E 1221 MOVQ R3,-(SP) : Save R3,R4,R5
044E 1222 PUSH R5 :
044E 1223
044E 1224 MOVZWL UCBSW_BCNT(R5),R3 :GET TRANSFER BYTE COUNT
044E 1225 MOVZWL UCBSW_BOFF(R5),R4 :GET BYTE OFFSET IN PAGE
044E 1226 MOVAB ^X3FF(R3)[R4],R3 :CALCULATE HIGHEST RELATIVE BYTE AND ROUND
044E 1227 ASHL #-9,R3,R3 :CALCULATE NUMBER OF MAP REGISTERS REQUIRED
044E 1228 5$:
044E 1229 MOVL UCBSL_CRB(R5),R1 : R1 => CRB
044E 1230 MOVL CRBSL_INTD+VECSL_ADP(R1),R2 : R2 => ADP
044E 1231 MOVAB CRBSL_INTD+VECSW_MAPREG(R1),R1 : R1 => UBMD.
044E 1232 BBS #VECSW_MAPLOCK,- : If SET, already permanently
044E 1233 UBMD$W_MAPREG(R1),30$ : allocated, so branch around.
044E 1234
044E 1235 INCL R3 : Round up request to next multiple
044E 1236 BICB #1,R3 : of 2.
044E 1237 MOVL ADPSL_MRACTMDRS(R2),R5 : R5 = index beyond last MRD.
044E 1238 BEQL 15$ : EQL implies no registers available.
044E 1239 10$:
044E 1240 CMPW R3,ADPSW_MRNREGARY-2(R2)[R5] : See if enough regs described here.
044E 1241 BLEQ 20$ : LEQ implies YES.
044E 1242
044E 1243 SOBGTR R5,10$ : Else branch back and continue
044E 1244 15$:
044E 1245 CLRL R0 : If here, allocation failure.
044E 1246 BRB 40$ : Branch around to return.
044E 1247 20$:
044E 1248 ADDW3 ADPSW_MRFREGARY-2(R2)[R5],- : Calculate register # beyond
044E 1249 ADPSW_MRNREGARY-2(R2)[R5],R0 : last extent.
044E 1250 SUBW R3,R0 : We allocate from high end. R0
044E 1251 : contains 1st reg. to alloc.
044E 1252 MOVW R0,UBMD$W_MAPREG(R1) : Record 1st register allocated.
```

7E 53 7D 044E 1216  
55 DD 0451 1217  
18 11 0453 1218  
7E 53 7D 0455 1221  
55 DD 0458 1222  
53 7E A5 3C 045A 1224  
54 7C A5 3C 045E 1225  
53 03FF C344 9E 0462 1226  
53 53 F7 8F 78 0468 1227  
51 24 A5 D0 046D 1228  
52 38 A1 D0 0471 1230  
51 34 A1 9E 0475 1231  
OF E0 0479 1232  
38 61 047B 1233  
53 D6 047D 1235  
53 01 8A 047F 1236  
55 5C A2 D0 0482 1237  
0A 13 0486 1238  
62 A245 53 B1 0488 1240  
07 15 048D 1241  
F6 55 F5 048F 1243  
50 D4 0492 1244  
22 11 0494 1246  
50 62 A245 015C C245 A1 0496 1248  
50 53 A2 049F 1249  
61 50 B0 04A2 1251  
04A2 1252



61	8000	8F	A8	04A5	1253	BISW	#VECSM_MAPLOCK,UBMDSW_MAPREG(R1);	and mark it permanent.
02	A1	53	90	04AA	1254	MOVB	R3,UBMDSB_NUMREG(R1)	; Set # of map regs allocated.
62	A245	53	A2	04AE	1255	SUBW	R3,ADPSW_MRNREGARY-2(R2)[R5]	; Subtract out # regs allocated.
		0A	13	0483	1256	BEQL	50\$	; EQL implies descriptor not
				0485	1257			; valid; branch to deallocate.
				0485	1258			
50	01	D0		0485	1259	MOVL	S^#SS\$ _NORMAL,R0	; Indicate success.
				0488	1260			
	55	8ED0		0488	1261	POPL	R5	; Restore R5,R4,R3
53	8E	7D		0488	1262	MOVQ	(SP)+,R3	;
		05		048E	1263	RSB		;
				048F	1264			
	55	D7		048F	1265	DECL	R5	; R5 = index of descriptor to dealloc.
0002	30			04C1	1266	BSBW	DEALLOC_DESCRIP	; Call to deallocate descriptor.
EF	11			04C4	1267	BRB	30\$	; And branch back to return.
				04C6	1268	.dsabl	lsl	

```
04C6 1270 :+
04C6 1271 : DEALLOC_DESCRIP - Common internal subroutine called to deallocate
04C6 1272 : a UBA Map Register descriptor.
04C6 1273 :
04C6 1274 : INPUTS:
04C6 1275 : R2 => ADP
04C6 1276 : R5 = index of descriptor to deallocate.
04C6 1277 : OUTPUTS:
04C6 1278 : The UBA Map Allocation structures are updated by contracting
04C6 1279 : descriptors over the deallocated one.
04C6 1280 : Register R5 is modified.
04C6 1281 :-
04C6 1282 :
04C6 1283 : DEALLOC_DESCRIP:
5C A2 D7 04C6 1284 : DECL ADPSL_MRACTMDRS(R2) ; Decrement # active descriptors.
64 A245 66 A245 B0 04C9 1285 10$:
015E C245 0160 C245 B0 04C9 1286 : MOVW ADPSW_MRNREGARY+2(R2)[R5],- ; Move data towards lower index
EB 55 5C A2 F2 04D0 1287 : ADPSW_MRNREGARY(R2)[R5] ; to fill up hole.
05 04D0 1288 : MOVW ADPSW_MRFREGARY+2(R2)[R5],-
04D9 1289 : ADPSW_MRFREGARY(R2)[R5]
04DE 1290 : AOBLS ADPSL_MRACTMDRS(R2),R5,10$ ; Loop thru rest of active MDRS.
04DF 1291 : RSB
04DF 1292 :
04DF 1293 :+
04DF 1294 : ALLOC_DESCRIP - Common internal subroutine to allocate a UBA map register
04DF 1295 : descriptor in the middle of the range of descriptors.
04DF 1296 :
04DF 1297 : INPUTS:
04DF 1298 : R2 => ADP
04DF 1299 : R5 = index of where we must allocate descriptor
04DF 1300 : OUTPUTS:
04DF 1301 : Allocation is accomplished by creating a hole in each of the arrays
04DF 1302 : by moving descriptor items to the next higher element.
04DF 1303 : Note R0 is modified.
04DF 1304 :-
04DF 1305 :
04DF 1306 : ALLOC_DESCRIP:
50 5C A2 D0 04DF 1307 : MOVL ADPSL_MRACTMDRS(R2),R0 ; R0 = # active descriptors.
55 50 D1 04E3 1308 10$:
64 A240 62 A240 B0 04E3 1309 : CMPL R0,R5 ; Have we finished?
015E C240 015C C240 B0 04E6 1310 : BLEQ 20$ ; LEQ implies YES.
EB 50 F5 04E8 1311 : MOVW ADPSW_MRNREGARY-2(R2)[R0],- ; Starting from ends of arrays,
5C A2 D6 04EF 1312 : ADPSW_MRNREGARY(R2)[R0] ; copy # register items.
05 04EF 1313 :
04F8 1314 : MOVW ADPSW_MRFREGARY-2(R2)[R0],-
04FB 1315 : ADPSW_MRFREGARY(R2)[R0]
04FB 1316 : SOBGTR R0,10$ ; And loop back until we reach
04FB 1317 : the hole we have created.
04FE 1318 20$: INCL ADPSL_MRACTMDRS(R2) ; Increment # active descriptors.
05 04FE 1319 : RSB ; Return to caller
```

```
04FF 1321 .SBTTL Release UNIBUS Map Registers
04FF 1322
04FF 1323 :+
04FF 1324 : IOC$RELMAPUDA - RELEASE UNIBUS MAP REGISTERS (CALLED FROM UDA PORT DRIVER)
04FF 1325 : IOC$RELMAPREG - RELEASE UNIBUS MAP REGISTERS
04FF 1326 : This routine is called to release UNIBUS map registers that were previously
04FF 1327 : assigned for an I/O transfer.
04FF 1328
04FF 1329 INPUTS:
04FF 1330 (For IOC$RELMAPUDA only)
04FF 1331
04FF 1332 R4 => PDT
04FF 1333 R5 => CDRP
04FF 1334
04FF 1335 (For IOC$RELMAPREG call only)
04FF 1336
04FF 1337 R5 = UCB ADDRESS OF DEVICE UNIT.
04FF 1338
04FF 1339 It is assumed that the caller still owns the I/O channel on which
04FF 1340 the transfer took place.
04FF 1341
04FF 1342 OUTPUTS:
04FF 1343
04FF 1344 : If the mapping registers have been permanently assigned to the asso-
04FF 1345 : ciated I/O channel (only possible for IOC$RELMAPREG), then control
04FF 1346 : is immediately returned to the caller. Else the mapping registers are
04FF 1347 : released (via a call to IOC$DALOCUBAMAP) and we then go into a loop
04FF 1348 : removing waiting driver processes from the Map Register Wait Queue
04FF 1349 : until either the Queue is completely drained or we run out of map
04FF 1350 : registers to satisfy the needs of a given waiting driver process.
04FF 1351 : Driver processes waiting here have their context stored in either
04FF 1352 : a UCB fork block or a CDRP fork block and the processing required to
04FF 1353 : resume each of these types of driver process is slightly different.
04FF 1354 : What is done for each is to allocate the required map registers
04FF 1355 : (via a call to IOC$ALOUBAMPA for UCB threads and via a call to
04FF 1356 : IOC$ALOUBAMAP for CDRP threads) and to resume the waiting driver
04FF 1357 : process. Resuming a UCB thread is done by restoring register
04FF 1358 : context and JSB'ing to the saved PC. Resuming a CDRP thread is
04FF 1359 : accomplished by calling SCSS$RESUMEWAITR.
04FF 1360 :-
04FF 1361 .enabl lsb
04FF 1362 IOC$RELMAPUDA::
04FF 1363 MOVQ R3,-(SP) ; Save R3-R6
04FF 1364 MOVQ R5,-(SP) ;
04FF 1365
04FF 1366 MOVL PDT$L_ADP(R4),R2 ; R2 => ADP.
04FF 1367 MOVL R2,R6 ; R6 => ADP also.
04FF 1368
04FF 1369 MOVAB CDRP$L_UBARSRC(R5),R3 ; R3 => UBMD.
04FF 1370 MOVZWL UBMD$L_MAPREG(R3),R4 ; R4 has 1st mapreg #.
04FF 1371 MOVZBL UBMD$L_NUMREG(R3),R3 ; R3 has # of mapregs.
04FF 1372 BRB 10$ ; Branch to common code.
04FF 1373
04FF 1374 IOC$RELMAPREG::
04FF 1375 MOVL UCB$L_CRB(R5),R1 ; Release unibus map registers
04FF 1376 BBS #VEC$L_MAPLOCK,- ; R1 => CRB.
04FF 1377 CRB$L_INTD+VEC$L_MAPREG(R1),50$ ; If SET, permanent allocation so branch.
```

7E	53	7D	04FF	1363	
7E	55	7D	0502	1364	
			0505	1365	
52	00E0	C4	DO	0505	1366
	56	52	DO	050A	1367
				050D	1368
53	3C	A5	9E	050D	1369
	54	63	3C	0511	1370
53	02	A3	9A	0514	1371
		1E	11	0518	1372
				051A	1373
				051A	1374
51	24	A5	DO	051A	1375
		0F	EO	051E	1376
3D	34	A1		0520	1377

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------



```
0573 1417 :+
0573 1418 : IOC$DALOCUBAMAP - Common internal subroutine to update the UBA Map allocation
0573 1419 : structures to include the map registers specified here among the
0573 1420 : available map registers.
0573 1421 :
0573 1422 : INPUTS:
0573 1423 : R2 => ADP
0573 1424 : R3 = # map registers to free.
0573 1425 : R4 = first map register to free.
0573 1426 :
0573 1427 : OUTPUTS:
0573 1428 : The UBA Map Allocation structures are updated.
0573 1429 :
0573 1430 : Registers R0, R1 and R5 are modified.
0573 1431 :
0573 1432 :-
0573 1433 :
0573 1434 : IOC$DALOCUBAMAP:
0573 1435 : CLRL R5 ; Initialize loop variable.
0573 1436 : ADDL3 R4,R3,R1 ; R1 = map register beyond extent.
0573 1437 : TSTL R3 ; Is the # of regs. to deallocate zero?
0573 1438 : BEQL 90$ ; Branch to bugcheck if zero.
0573 1439 : TSTL ADP$L_MRACTMDRS(R2) ; Test for zero active descriptors.
0573 1440 : BEQL 50$ ; EQL implies no registers available.
0573 1441 :
0573 1442 : 10$:
0573 1443 : CMPW R1,ADP$W_MRFREGARY(R2)[R5] ; See if map registers to free
0573 1444 : ; are before those described
0573 1445 : ; by current descriptor.
0573 1446 : BLEQ 20$ ; LEQ implies yes.
0573 1447 :
0573 1448 : AOBLS ADP$L_MRACTMDRS(R2),R5,10$ ; Else branch back and try next.
0573 1449 : BRB 40$ ; If here, registers to free
0573 1450 : ; beyond those described by
0573 1451 : ; last descriptor. So branch
0573 1452 : ; to try and absorb at end of
0573 1453 : ; last descriptor.
0573 1454 :
0573 1455 : 20$:
0573 1456 : BNEQ 40$ ; NEQ implies that although we alloca-
0573 1457 : ; registers before the current des-
0573 1458 : ; criptor, we are not contiguous with
0573 1459 : ; it. So we branch to try and absorb
0573 1460 : ; these registers in the previous one.
0573 1461 :
0573 1462 : ; Here we can absorb the registers in the current descriptor.
0573 1463 :
0573 1464 : ADDW3 ADP$W_MRNREGARY-2(R2)[R5],- ; Calculate end of previous
0573 1465 : ADP$W_MRFREGARY-2(R2)[R5],R0 ; extent and move to R0.
0573 1466 : CMPW R0,R4 ; Does it coincide with start
0573 1467 : ; of this extent?
0573 1468 : BEQL 30$ ; EQL implies yes.
0573 1469 :
0573 1470 : ; Here we have the most likely case. The map registers that we are freeing can
0573 1471 : ; be absorbed into the top of the current descriptor but not also in the
0573 1472 : ; previous descriptor.
0573 1473 :
0573 1474 : MOVW R4,ADP$W_MRFREGARY(R2)[R5] ; First register freed becomes
0573 1475 : ; first register of current
```

51 53 55 D4 0573 1435  
54 C1 0573 1436  
53 D5 0573 1437  
62 13 0573 1438  
5C A2 D5 0573 1439  
4E 13 0580 1440  
015E C245 51 B1 0582 1441  
0588 1442  
0588 1443  
07 15 0588 1444  
0588 1445  
F3 55 5C A2 F2 058A 1446  
2B 11 058A 1447  
058F 1448  
0591 1449  
0591 1450  
0591 1451  
0591 1452  
0591 1453  
29 12 0591 1454  
0593 1455  
0593 1456  
0593 1457  
0593 1458  
0593 1459  
0593 1460  
0593 1461  
50 015C C245 62 A245 A1 0593 1462  
059C 1463  
54 50 B1 059C 1464  
059F 1465  
0C 13 059F 1466  
05A1 1467  
05A1 1468  
05A1 1469  
05A1 1470  
05A1 1471  
015E C245 54 B0 05A1 1472  
05A7 1473

```
64 A245 53 A0 05A7 1474      ADDW  R3,ADPSW_MRNREGARY(R2)[R5]      ; descriptor.
                                05A7 1475      ; Number of registers is sum of
                                05AC 1476      ; registers freed and registers
                                05AC 1477      ; previously described here.
                                05      1478      RSB
                                05AD 1479
                                05AD 1480 ; Here we have the case where the map registers being freed fall between two
                                05AD 1481 ; discontinuous blocks and exactly span the difference. We then can
                                05AD 1482 ; describe the entire group with one descriptor, and so we also
                                05AD 1483 ; deallocate the current descriptor. Note new combined descriptor
                                05AD 1484 ; will still begin at same map register number so we do NOT alter
                                05AD 1485 ; this item.
                                05AD 1486
                                05AD 1487 30$:
62 A245 53 A0 05AD 1488      ADDW  R3,ADPSW_MRNREGARY-2(R2)[R5]      ; Partial sum of registers
                                05B2 1489      ; being freed and previous ones.
62 A245 64 A245 A0 05B2 1490      ADDW  ADPSW_MRNREGARY(R2)[R5],-      ; Now add in registers described
                                05B9 1491      ADPSW_MRNREGARY-2(R2)[R5]      ; in current descriptor.
                                05B9 1492
                                FFOA 31 05B9 1493      BRW  DEALLOC_DESCRIP      ; BRW to subroutine and let it
                                05BC 1494      ; return to our caller.
                                05BC 1495
                                05BC 1496 ; Here we cannot absorb the freed map registers in the current descriptor.
                                05BC 1497 ; We test to see if we can absorb them in the previous descriptor.
                                05BC 1498
                                05BC 1499 40$:
50 015C C245 62 A245 A1 05BC 1500      ADDW3  ADPSW_MRNREGARY-2(R2)[R5],-      ; Calculate end of previous
                                05C5 1501      ADPSW_MRFREGARY-2(R2)[R5],R0      ; extent and move to R0.
                                54 50 B1 05C5 1502      CMPW  R0,R4      ; See if contiguous with previous.
                                06 12 05C8 1503      BNEQ  50$      ; NEQ implies NO.
                                05CA 1504
                                62 A245 53 A0 05CA 1505      ADDW  R3,ADPSW_MRNREGARY-2(R2)[R5]      ; Sum # of registers in extent.
                                05      1506      RSB
                                05D0 1507
                                05D0 1508 ; Here we must allocate a new descriptor to describe the map registers we
                                05D0 1509 ; are freeing. Conditions at this time are as follows:
                                05D0 1510
                                05D0 1511      R2 => ADP
                                05D0 1512      R3 = # registers to free
                                05D0 1513      R4 = first register to free
                                05D0 1514      R5 = index of where we must allocate descriptor
                                05D0 1515
                                05D0 1516      Allocation is accomplished by calling subroutine ALLOC_DESCRIP
                                05D0 1517
                                05D0 1518
                                05D0 1519 50$:
                                64 A245 FFOC 30 05D0 1520      BSBW  ALLOC_DESCRIP      ; Alloc R5 = index of descriptor.
                                015E C245 53 B0 05D3 1521      MOVW  R3,ADPSW_MRNREGARY(R2)[R5]      ; Fill in allocated descriptor.
                                54 B0 05D8 1522      MOVW  R4,ADPSW_MRFREGARY(R2)[R5]
                                05 05DE 1523      RSB
                                05DF 1524
                                05DF 1525 90$:      BUG_CHECK INCONSTATE      ; Non-fatal bugcheck on zero map
                                05E3 1526      ; registers deallocation attempts.
                                05      1527      RSB      ; Then ignore deallocate request.
```

```

05E4 1529      .SBTTL RETURN TO CALLER
05E4 1530      :+
05E4 1531      : IOC$RETURN - RETURN TO CALLER
05E4 1532      :
05E4 1533      : THIS ROUTINE IS CALLED AS A RESULT OF A DDT DISPATCH TO A NULL ENTRY. ITS
05E4 1534      : FUNCTION IS MERELY TO RETURN TO ITS CALLER.
05E4 1535      :
05E4 1536      : INPUTS:
05E4 1537      :
05E4 1538      :     NONE.
05E4 1539      :
05E4 1540      : OUTPUTS:
05E4 1541      :
05E4 1542      :     NONE.
05E4 1543      : -
05E4 1544      :
05E4 1545      : IOC$RETURN::
05E4 1546      : RSB
                                : RETURN TO CALLER
                                :

```

```
05E5 1548 .SBTTL WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
05E5 1549
05E5 1550 :+ IOCSWFIKPCB - WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
05E5 1551
05E5 1552 : THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON
05E5 1553 : A DEVICE UNIT AND TO KEEP THE CHANNEL. THIS ROUTINE CAN BE CALLED AT
05E5 1554 : EITHER FORK OR DEVICE INTERRUPT LEVEL.
05E5 1555
05E5 1556 INPUTS:
05E5 1557
05E5 1558 00(SP) = RETURN ADDRESS OF CALLER.
05E5 1559 04(SP) = TIMEOUT VALUE IN SECONDS.
05E5 1560 08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
05E5 1561 12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
05E5 1562
05E5 1563 R5 = UCB ADDRESS OF DEVICE UNIT.
05E5 1564
05E5 1565 OUTPUTS:
05E5 1566
05E5 1567 THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
05E5 1568 R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
05E5 1569 TIMEOUT ARE ENABLED, AND A RETURN TO THE CALLER'S CALLER IS EXECUTED.
05E5 1570 :-
05E5 1571
05E5 1572 IOCSWFIKPCB::
05E5 1573 ADDL #2,(SP) ;WAITFOR INTERRUPT/TIMEOUT AND KEEP CHANNEL
05E8 1574 MOVQ R3,UCB$L_FR3(R5) ;CALCULATE OFFSET TO NORMAL RETURN
05EC 1575 POPL UCB$L_FPC(R5) ;SAVE REGISTERS R3 AND R4
05F0 1576 BISM #UCB$M_INT,UCB$M_TIM,UCB$M_STS(R5) ;SAVE INTERRUPT RETURN ADDRESS
05F4 1577 ADDL3 (SP)+,["EXESGL ABSTIM,UCB$C_DUETIME(R5) ;ENABLE INTERRUPT AND TIMEOUT
05FD 1578 BICW #UCB$M_TIMEOUT,UCB$M_STS(R5) ;SET TIMEOUT TIME
0603 1579 ENBINT ;CLEAR UNIT TIMED OUT
0606 1580 RSB ;ENABLE INTERRUPTS
;
```

6E 02 C0  
10 A5 53 7D  
OC A5 8ED0  
64 A5 03 A8  
6C A5 00000000 EF 8E C1  
64 A5 0040 8F AA  
05 0603 1579  
05 0606 1580



```
0607 1582 .SBTTL WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
0607 1583
0607 1584 IOC$WFIRLCH - WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
0607 1585
0607 1586 THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON A DEVICE
0607 1587 UNIT AND TO RELEASE THE CHANNEL. THIS ROUTINE CAN ONLY BE CALLED AT FORK LEVEL.
0607 1588
0607 1589 INPUTS:
0607 1590
0607 1591 00(SP) = RETURN ADDRESS OF CALLER.
0607 1592 04(SP) = TIMEOUT VALUE IN SECONDS.
0607 1593 08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
0607 1594 12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
0607 1595
0607 1596 R5 = UCB ADDRESS OF DEVICE UNIT.
0607 1597
0607 1598 OUTPUTS:
0607 1599
0607 1600 THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
0607 1601 R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
0607 1602 TIMEOUT ARE ENABLED, THE CHANNEL IS RELEASED, AND A RETURN TO THE CALLER'S
0607 1603 CALLER IS EXECUTED.
0607 1604 :-
0607 1605
0607 1606 IOC$WFIRLCH::
0607 1607 ADDL #2,(SP) ;WAITFOR INTERRUPT/TIMEOUT AND RELEASE CHANN
060A 1608 MOVQ R3,UCB$L FR3(R5) ;CALCULATE OFFSET TO NORMAL RETURN
060E 1609 POPL UCB$L FPC(R5) ;SAVE REGISTERS R3 AND R4
0612 1610 BICW #UCB$M_INT!UCB$M_TIM,UCB$W_STS(R5) ;SAVE INTERRUPT RETURN ADDRESS
0616 1611 ADDL3 (SP)+,C^EXESGL ABSTIM,UCB$C_DUETIM(R5) ;ENABLE INTERRUPT AND TIMEOUT
061F 1612 BICW #UCB$M_TIMEOUT,UCB$W_STS(R5) ;SET TIMEOUT TIME
0625 1613 ENBINT ;CLEAR UNIT TIMED OUT
0628 1614 BRW IOC$RELCHAN ;ENABLE INTERRUPTS
062B 1615 ;RELEASE ALL CHANNELS AND RETURN TO CALLER
062B 1616
```

```

      6E 02 C0
    10 A5 53 7D
      0C A5 8E D0
    64 A5 03 A8
6C A5 00000000'EF 8E C1
    64 A5 0040 8F AA
      FA5F 31
```

```
062B 1618 .SBTTL ALLOCATE SYSTEM PAGE TABLE
062B 1619 :+
062B 1620 : IOC$ALLOSPT - ALLOCATE SYSTEM PAGE TABLE
062B 1621 :
062B 1622 : THIS ROUTINE ALLOCATES SYSTEM PAGE TABLE (SPT) ENTRIES.
062B 1623 :
062B 1624 : INPUTS:
062B 1625 :
062B 1626 : R1 = NUMBER OF SPT ENTRIES TO BE ALLOCATED
062B 1627 :
062B 1628 : BOO$GL_SPTFREL = LOWEST FREE VPN
062B 1629 : BOO$GL_SPTFRELH = HIGHEST FREE VPN
062B 1630 :
062B 1631 : IT IS ASSUMED THAT THE CALLER IS RUNNING AT IPL$_SYNCH.
062B 1632 :
062B 1633 : OUTPUTS:
062B 1634 :
062B 1635 : R0 = SUCCESS INDICATION.
062B 1636 : R2 = STARTING PAGE NUMBER ALLOCATED (SVPN).
062B 1637 : R3 = ADDRESS OF BASE OF SYSTEM PAGE TABLE (MMG$GL_SPTBASE).
062B 1638 :
062B 1639 : R1 IS PRESERVED ACROSS CALL.
062B 1640 :
062B 1641 : IOC$ALLOSPT::
062B 1642 : CLRL R0 ;ALLOCATE SYSTEM PAGE TABLE
062B 1643 : MOVL L^BOO$GL_SPTFREL,R2 ;ASSUME FAILURE
062B 1644 : ADDL3 R1,R2,R3 ;GET NEXT AVAILABLE SYSTEM VPN
062B 1645 : CMPL R3,L^BOO$GL_SPTFRELH ;COMPUTE NEXT WITH THIS ALLOCATION
062B 1646 : BGEQU 10$ ;ARE THERE ENOUGH AVAILABLE?
062B 1647 : MOVL R3,L^BOO$GL_SPTFREL ;BR IF NO
062B 1648 : MOVL L^MMG$GL_SPTBASE,R3 ;MARK THE ENTRIES ALLOCATED
062B 1649 : INCL R0 ;GET ADDR OF BASE OF SPT
062B 1650 : 10$: ;SET SUCCESS
062B 1651 : RSB ;
```

52	00000000'EF	50	D4	062B	1642
53	52	51	D0	062D	1643
00000000'EF	53	D1	C1	0634	1644
	10	1E	D1	0638	1645
00000000'EF	53	D0	063F	1646	
53	00000000'EF	50	D0	0641	1647
			D0	0648	1648
			D6	064F	1649
			05	0651	1650
				0651	1651

```
0652 1653 .SBTTL CONVERT DEVICE NAME AND UNIT
0652 1654
0652 1655 + IOC$CVT_DEVNAM - Convert device name and unit
0652 1656
0652 1657 This routine is called to convert a device name and unit number to a physical
0652 1658 device name string.
0652 1659
0652 1660 Inputs:
0652 1661
0652 1662 The caller is assumed to have PROBED the output buffer for write access.
0652 1663 The I/O data base is locked for read access. This means that the caller
0652 1664 owns the I/O data base mutex and/or is at IPL SYNCH or higher.
0652 1665
0652 1666 R0 = length of output buffer.
0652 1667 R1 = address of output buffer.
0652 1668 R4 = name string formation mode, one of:
0652 1669 -1 (DVIS_DEVNAM) -- a name suitable for displays
0652 1670 for non-local devices, return node$ddcn
0652 1671 for local devices:
0652 1672 if in cluster and file oriented device, return node$ddcn
0652 1673 otherwise, return ddcn
0652 1674 0 (DVIS_FULLDEVNAM) -- a name with appropriate node information
0652 1675 if allocation class not zero and file oriented device, return
0652 1676 $allocclass$ddcn
0652 1677 otherwise, return node$ddcn
0652 1678 1 (DVIS_ALLDEVNAM) -- a name with allocation class information
0652 1679 if allocation class not zero, return $allocclass$ddcn
0652 1680 otherwise, return node$ddcn
0652 1681 2 (no GETDVI item code) -- an old fashioned name
0652 1682 return ddcn
0652 1683 3 (no GETDVI item code) -- a secondary path name for displays
0652 1684 same as -1 except secondary path name returned
0652 1685 4 (no GETDVI item code) -- path controller name for displays
0652 1686 same as -1 except no unit number is appended
0652 1687 Note: if the node name string is null, node$ is not returned.
0652 1688 R5 = address of device UCB.
0652 1689
0652 1690 Outputs:
0652 1691
0652 1692 The device name and unit number are converted and stored in the specified
0652 1693 output buffer. The following register values are returned:
0652 1694
0652 1695 R0 = Final conversion status.
0652 1696 SSS_NORMAL or
0652 1697 SSS_BUFFEROVF (an alternate success status which
0652 1698 indicates that the supplied buffer could not
0652 1699 hold the device name string)
0652 1700 R1 = Length of conversion string. R1 = 0 if the alternate
0652 1701 path name was requested but none exists.
0652 1702
0652 1703
0652 1704
0652 1705 Working storage (offsets from R7)
0652 1706
0652 1707 $OFFSET 0, POSITIVE, < -
0652 1708 <BINNUM, 8>, - ; Binary value to convert to ASCII
0652 1709 - ; add new working storage cells before this line
```

```
0652 1710 <RESR0>, - ;Result R0
0652 1711 <RESR1>, - ;Result R1
0652 1712 <SCRLN,0> - ;amount of working storage
0652 1713 <RESR2>, - ;saved R2
0652 1714 <RESR3>, - ;saved R3
0652 1715 <RESR4>, - ;saved R4
0652 1716 >
0000 BINNUM:
0008 RESR0:
000C RESR1:
0010 SCRLN:
0010 RESR2:
0014 RESR3:
0018 RESR4:
0652 1717
0652 1718 IOC$CVT_DEVNAM:: ;Convert device name and unit
0652 1719
00FC 8F BB 0652 1720 PUSHR #M<R2,R3,R4,R5,R6,R7> ;Save registers
0656 1721 :
0656 1722 : Push a quadword onto the stack. The quadword will land
0656 1723 : on the stack so that when the POPR at the end of the routine
0656 1724 : is executed, R0 will contain the routine value, and R1 will
0656 1725 : contain the length of the formatted device name.
0656 1726 :
7E 01 7D 0656 1727 MOVQ #SS$ NORMAL,-(SP) ;Put a 1 and a 0 on the stack
7E 7C 0659 1728 CLRQ -(SP) ;Init binary number working area.
57 5E D0 065B 1729 ASSUME SCRLN EQ 16
065E 1730 MOVL SP, R7 ;Setup result R0 and R1 pointer in R7.
065E 1731 :
065E 1732 : Precede the device name with a "_" (underscore character) to
065E 1733 : indicate that this is a physical device name.
065E 1734 :
53 5F 8F 9A 065E 1735 MOVZBL #A/ /,R3 ;Put underscore character in R3
00B4 30 0662 1736 BSBW PUTCHAR ;Put it in the output buffer
0665 1737 :
0665 1738 : Check for a possible nodename. If it exists, determine which format
0665 1739 : of name was requested by the caller.
0665 1740 :
56 28 A5 D0 0665 1741 MOVL UCBSL_DDB(R5),R6 ;Get DDB address
52 34 A6 D0 0669 1742 MOVL DDBSL_SB(R6),R2 ;Get System Block address
5D 13 066D 1743 BEQL LOCAL_NAME ;None, leave
09 E1 066F 1744 BBC #DEV$V_NNM,- ;Branch if nodename not wanted
58 3C A5 0671 1745 UCBSL_DEVCHAR2(R5),LOCAL_NAME
0674 1746 CASE R4,- ;Dispatch on type of output requested:
0674 1747 limit=#-1,displist=<-
0674 1748 DISPLAY_NAME,- ; -1 ==> node$dev: for disks, else dev:
0674 1749 FULL_NAME,- ; 0 ==> $allocs$dev: or node$dev:
0674 1750 ALLOC_NAME,- ; 1 ==> $allocs$dev: or node$dev:
0674 1751 LOCAL_NAME,- ; 2 ==> just dev:
0674 1752 SECONDARY_NAME,- ; 3 ==> secondary path
0674 1753 DISPLAY_NAME - ; 4 ==> same as -1 sans unit number
0674 1754 >
5B 11 0686 1755 BRB EXDVNM ; All others are NOPs.
0688 1756
0688 1757 FULL_NAME:
33 38 A5 0E E1 0688 1758 BBC #DEV$V_FOD,- ;A file oriented device?
068D 1759 UCBSL_DEVCHAR(R5),-
```



```
068D 1760 ADD_NODE ;Branch if not file oriented device.
068D 1761
068D 1762 ALLOC_NAME:
068D 1763
67 3C A6 9A 068D 1764 MOVZBL DDB$L_ALLOCLS(R6), - ;Setup allocation class value
0691 1765 BINNUM(R7) ; for conversion.
2D 13 0691 1766 BEQL ADD_NODE ;If none return node+device name.
0080 30 0693 1767 BSBB PUTDOLLAR ;Prepend allocation class with a '$'
58 10 0696 1768 BSBB PUTNUM ;Convert allocation class number to
30 11 0698 1769 ;ASCII and put it in the buffer
0698 1770 BRB ADD_DOLLAR ;Append dollar sign to alloc. class
069A 1771 ; and add device name to buffer.
069A 1772
069A 1773 SECONDARY_NAME:
3C A5 E1 069A 1774 BBC #DEV$V_2P, - ;Branch if device not dual-pathed.
069C 1775 UCBSL_DEVCHAR2(R5), - ; (I.E. there is no secondary path to
4C 069E 1776 NO_SECONDARY ; return.)
56 00A0 C5 D0 069F 1777 MOVL UCBSL_DP_DDB(R5),R6 ;Get secondary DDB.
45 13 06A4 1778 BEQL NO_SECONDARY ;Branch to no sec. path if none.
52 34 A6 D0 06A6 1779 MOVL DDB$L_SB(R6),R2 ;Get alternate SB.
06AA 1780
06AA 1781 DISPLAY_NAME:
06AA 1782 CMPL R2,#SCSSGA_LOCALSB ;Is it the perm local system block?
0D 12 06B1 1783 BNEQ ADD_NODE ;Return node+devnam for non-local devs.
06B3 1784 IFNOCLSTR LOCAL_NAME ;Return devnam if not part of a cluster.
0C 38 A5 0E E1 06BB 1785 BBC #DEV$V_FOD, - ;A file oriented device?
06C0 1786 UCBSL_DEVCHAR(R5), -
06C0 1787 LOCAL_NAME ;Branch if not a file oriented device.
06C0 1788 ;Its a local disk in a cluster: return
06C0 1789 ;node+device name format.
06C0 1790
06C0 1791 ; Return node name, plus device name. Copy node name to buffer and
06C0 1792 ; suffix with a '$' before moving in rest of device name.
06C0 1793
06C0 1794 ADD_NODE:
52 44 A2 9E 06C0 1795 MOVAB SB$T_NODENAME(R2),R2 ;Point to name field
62 95 06C4 1796 TSTB (R2) ;Is the node name null?
04 13 06C6 1797 BEQL LOCAL_NAME ;Skip inserting node name, if its null.
3E 10 06C8 1798 BSBB PUTASCII ;Copy counted ASCII str. to output buf.
4A 10 06CA 1799 ADD_DOLLAR: BSBB PUTDOLLAR ;Append dollar sign to node name
06CC 1801
06CC 1802 ; Copy device name to buffer.
06CC 1803
06CC 1804 LOCAL_NAME:
52 14 A6 9E 06CC 1805 MOVAB DDB$T_NAME(R6),R2 ;Get address of ASCII device name.
36 10 06D0 1806 BSBB PUTASCII ;Copy counted ASCII str. to output buf.
04 18 A7 B1 06D2 1807 CMPW RESR4(R7),#4 ;Do we want the unit number?
0B 13 06D6 1808 BEQL EXDVNM ;Nope
67 54 A5 3C 06D8 1809 MOVZWL UCBSW_UNIT(R5), - ;Setup device unit number for
06DC 1810 BINNUM(R7) ; conversion to ASCII.
12 10 06DC 1811 BSBB PUTNUM ;Convert unit number to ASCII.
06DE 1812
06DE 1813 ; Terminate the device name with a ':' (colon).
06DE 1814
53 3A 9A 06DE 1815 MOVZBL #*A/://,R3 ;Put a ':' in R3
36 10 06E1 1816 BSBB PUTCHAR ;Put the ':' in output buffer
```

```
06E3 1817 :  
06E3 1818 : Clean up the stack and exit. The stack has been set up so that  
06E3 1819 : the proper values will be stored in R0 and R1 by the POPR.  
06E3 1820 :  
5E 08 C0 06E3 1821 EXDVM: ADDL #RESR0,SP ;Remove everything upto result R0  
06E6 1822 : from the stack  
00FF 8F BA 06E6 1823 POPR #^M<R0,R1,R2,R3,R4,R5,R6,R7> ;Restore registers  
05 06EA 1824 RSB ;Return  
06EB 1825 :  
06EB 1826 :  
06EB 1827 : Come here when the secondary device name was requested but none exists.  
06EB 1828 :  
06EB 1829 NO_SECONDARY:  
OC A7 D4 06EB 1830 CLRL RESR1(R7) ;Clear count of characters  
F3 11 06EE 1831 BRB EXDVM ;and return.  
06F0 1832 :  
06F0 1833 :  
06F0 1834 :++  
06F0 1835 : The following code is a local subroutine to convert binary to ASCII and  
06F0 1836 : put the ASCII equivalent in the output name buffer.  
06F0 1837 :  
06F0 1838 Inputs:  
06F0 1839 :  
06F0 1840 BINNUM(R7) binary number to be converted (a quadword with high  
06F0 1841 : longword zeroed  
06F0 1842 :  
06F0 1843 Outputs:  
06F0 1844 : The number at BINNUM(R7) is converted to ASCII and stored in the  
06F0 1845 : device name buffer.  
06F0 1846 :--  
06F0 1847 PUTNUM:  
53 01 8E 06F0 1848 MNEGB #1, R3 ;Get end-of-number marker.  
7E 53 90 06F3 1849 10$: MOVB R3, -(SP) ;Move digit/marker to scratch.  
53 67 67 0A 7B 06F6 1850 EDIV #10, BINNUM(R7), - ;Divide number by 10, overwrite number  
F6 12 06FB 1851 BINNUM(R7), R3 ;with quotient, put remainder in R3.  
06FD 1852 BNEQ 10$ ;If quotient not zero, go save this  
06FD 1853 : digit and get the next one.  
06FD 1854 :  
06FD 1855 : Get digits -- most significant first (then saved ones), convert them to  
06FD 1856 : ASCII, and put them in the output buffer  
06FD 1857 :  
53 30 80 06FD 1858 50$: AADB #^A/O/, R3 ;Convert binary digit to ASCII  
17 10 0700 1859 BSBB PUTCHAR ;Copy digit to output buffer  
53 8E 90 0702 1860 MOVB (SP)+, R3 ;Get another digit  
F6 18 0705 1861 BGEQ 50$ ;Branch if the end  
05 0707 1862 RSB  
0708 1863 :  
0708 1864 :++  
0708 1865 : The following code is a local subroutine to copy a counted ASCII string  
0708 1866 : to the output name buffer.  
0708 1867 :  
0708 1868 Inputs:  
0708 1869 :  
0708 1870 R2 Beginning address of a counted ASCII string  
0708 1871 :  
0708 1872 Outputs:  
0708 1873 : The counted ASCII string pointed to by R2 is copied to the device
```

```
0708 1874 :-- name buffer.
0708 1875 :--
0708 1876 PUTASCIC:
54 82 9A 0708 1877 MOVZBL (R2)+, R4 ;Get counted string length.
08 13 0708 1878 BEQL 90$ ;If no characters, leave.
53 82 90 0700 1879 $$: MOVBL (R2)+, R3 ;Move one byte to output buffer.
07 10 0710 1880 BSBB PUTCHAR ;Put the character in the output buffer.
F8 54 F5 0712 1881 SOBGTR R4, $$ ;Branch if more to copy.
05 0715 1882 90$: RSB ;All done, return.
0716 1883
0716 1884 :++
0716 1885
0716 1886 :The following code is a local subroutine to place a given
0716 1887 :byte in the output buffer. A count is kept of all characters
0716 1888 :placed in the output buffer. If the output buffer is full,
0716 1889 :the byte is not copied, the count is not increased, and the
0716 1890 :return status for IOC$CVT_DEVNAM is changed to SS$_BUFFEROVF
0716 1891 : (an alternate success status).
0716 1892
0716 1893 :Inputs:
0716 1894 :R0 Count of unstored character slots remaining in output buffer
0716 1895 :R1 Address of next unused character slot in output buffer
0716 1896 :R3 Character to be placed in the buffer
0716 1897
0716 1898 :Implicit inputs:
0716 1899 :RESR0(R7) longword holding final IOC$CVT_DEVNAM status
0716 1900 :RESR1(R7) longword holding final IOC$CVT_DEVNAM count of
0716 1901 : characters stored in the buffer (to be
0716 1902 : returned in R1
0716 1903
0716 1904 :Outputs:
0716 1905 :None.
0716 1906
0716 1907 :Implicit outputs:
0716 1908 :If R0 >= zero:
0716 1909 :R0 <== R0 - 1
0716 1910 : (R1) <== R3
0716 1911 :R1 <== R1 + 1
0716 1912 :RESR1(R7) <== RESR1(R7) + 1
0716 1913 :otherwise:
0716 1914 :RESR0(R7) <== SS$_BUFFEROVF
0716 1915 :++
0716 1916 :PUTDOLLAR is an internal routine which is the equivalent of:
0716 1917
0716 1918 :MOVBL #^A/$/, R3
0716 1919 :BSBB PUTCHAR
0716 1920 :--
0716 1921 :PUTDOLLAR:
53 24 90 0716 1922 :MOVBL #^A/$/, R3 ;Setup to put '$' in output buffer.
0719 1923 :PUTCHAR:
0719 1924 :DECL R0 ;Decrease characters remaining count.
071B 1925 :BLSS 90$ ;Branch if no more characters remaining.
81 53 90 0710 1926 :MOVBL R3, (R1)+ ;Copy character to output buffer
OC A7 D6 0720 1927 :INCL RESR1(R7) ;Count characters stored
05 0723 1928 :RSB ;Return
0724 1929
08 A7 0601 8F 3C 0724 1930 90$: MOVZWL #SS$_BUFFEROVF, - ;Set buffer overflow status
```

IOSUBNPAG  
V04-000

- NONPAGED I/O RELATED SUBROUTINES<sup>E</sup> 6  
CONVERT DEVICE NAME AND UNIT

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 45  
(27)

05 072A 1931 RESRO(R7)  
072A 1932 RSB



```
072B 1934 .SBTTL BROADCAST TO A TERMINAL
072B 1935 ++
072B 1936 IOC$BROADCAST
072B 1937
072B 1938 This routine will allow driver fork processes to broadcast a
072B 1939 given message to given terminal. The broadcast request is
072B 1940 dispatched to the proper terminal and control returns immediately
072B 1941 to the caller. Some time later the broadcast will complete, and
072B 1942 at that time all the necessary post-processing will be done.
072B 1943
072B 1944 This routine does not implement all the features of the $BRDCST system
072B 1945 service, but only the bare minimum necessary to send a message to a
072B 1946 single terminal. For more information about the terminal broadcast
072B 1947 mechanism, see the module SYSBRDCST.
072B 1948
072B 1949 Input:
072B 1950
072B 1951 R1 = Message length
072B 1952 R2 = Message address
072B 1953 R5 = Address of target terminal's UCB
072B 1954
072B 1955 Implicit input:
072B 1956
072B 1957 IPL$_ASTDEL <= CURRENT_IPL <= UCB$_FIPL(R5)
072B 1958
072B 1959 Output:
072B 1960
072B 1961 None. The contents of R1 .. R5 are preserved across the call.
072B 1962
072B 1963 Routine value:
072B 1964
072B 1965 SSS_NORMAL - The broadcast completed successfully.
072B 1966 SSS_INSMEM - Insufficient dynamic nonpaged pool for the request.
072B 1967 SSS_DEVOFFLINE - The target terminal has rejected the request.
072B 1968 SSS_ILLIOFUNC - The specified UCB does not belong to a terminal.
072B 1969 (Therefore a BROADCAST is an illegal I/O function.)
072B 1970 --
072B 1971
00000000 072B 1972 SAVED_R0 = 0
00000004 072B 1973 SAVED_R1 = 4
00000008 072B 1974 SAVED_R2 = 8
0000000C 072B 1975 SAVED_R3 = 12
00000010 072B 1976 SAVED_R4 = 16
00000014 072B 1977 SAVED_R5 = 20
072B 1978
072B 1979 IOC$BROADCAST::
072B 1980 MOVZWL #SS$_ILLIOFUNC,R0 ; Broadcast to a terminal
072B 1981 BBC #DEV$V TRM,- ; Assume device not a terminal
072B 1982 UCB$_DEVCHAR(R5),14$ ; Branch if not a terminal
072B 1983 PUSHR #^M<R0,R1,R2,R3,R4,R5> ; Save R0 .. R5
072B 1984 ADDL2 #TTY$K WB LENGTH,R1 ; Calculate the total pool required
072B 1985 MOVZWL #SS$_INSMEM,SAVED_R0(SP) ; Assume allocation failure
072B 1986 BSBW EX$ALONONPAGED ; Allocate the pool
072B 1987 BLBC R0,13$ ; Exit if error
072B 1988
072B 1989 ; Fill in the Terminal Write Packet (TWP).
072B 1990 ; Note that EX$ALONONPAGED the pool size
```

50 00F4 8F 3C  
56 38 A5 E1  
51 30 BB  
6E 0124 8F 3C  
F8BE 30  
44 50 E9

Symbolic offsets to saved registers

```
08 A2 51 B0 0745 1991 ; in R1 and the pool address in R2.
      30 90 0745 1992 ;
      0A A2 90 0745 1993 MOVW R1,TTY$WB_SIZE(R2) ; Set TWP size
      06 90 0749 1994 MOVW #DYN$C_TWP,= ; Set TWP structure type
      0B A2 90 074B 1995 MOVW TTY$WB_TYPE(R2) ;
      10 A2 01 D0 074D 1995 MOVW #IPL$-QUEUEAST,- ; Set the TWP fork IPL (for later use)
      30 A2 9E 074F 1997 MOVW TTY$WB_FIPL(R2) ;
      1C A2 9E 0751 1998 MOVL #1,TTY$WB_FR3(R2) ; Request refresh of read prompt
      04 AE C1 0755 1999 MOVAB TTY$WB_DATA(R2),- ; Set address of message start
      1C A2 9E 0758 2000 MOVAB TTY$WB_NEXT(R2) ;
      20 A2 9E 075A 2001 ADDL3 SAVED_R1(SP),- ; Set address of message end
      96 AF 9E 075D 2002 MOVAB TTY$WB_NEXT(R2),- ;
      2C A2 9E 075F 2003 MOVAB TTY$WB_END(R2) ;
      24 A2 D4 0761 2004 MOVAB B*END_BROADCAST,- ; Set callback address
      52 DD 0764 2005 CLRL TTY$WB_RETADDR(R2) ;
      08 AE 28 0766 2006 PUSHL R2 ; Clear pointer to associated IRP
      0C BE 28 0769 2007 MOVCL 4+SAVED_R1(SP),- ; Save TWP address
      30 A2 28 076B 2008 MOVCL 24+SAVED_R2(SP),- ; Copy the message text to the TWP
      30 A2 28 076E 2009 MOVCL TTY$WB_DATA(R2) ; (note the stack depth changed)
      30 A2 28 0770 2010 ;
      30 A2 28 0772 2011 ;
      30 A2 28 0772 2012 ; Queue the broadcast request to the terminal.
      30 A2 28 0772 2013 ; The disposition of the broadcast request will be determined
      30 A2 28 0772 2014 ; by the contents of TTY$WB_END. Note that if the request is
      30 A2 28 0772 2015 ; accepted by a remote terminal, or is rejected outright, the
      30 A2 28 0772 2016 ; TWP is no longer needed, and may be deallocated. The TTY$WB_END
      30 A2 28 0772 2017 ; field of the TWP will contain one of the following values:
      30 A2 28 0772 2018 ;
      30 A2 28 0772 2019 ; System address: request accepted by TTDRIVER
      30 A2 28 0772 2020 ; 1: request accepted by RTTDRIVER
      30 A2 28 0772 2021 ; 2: request rejected
      30 A2 28 0772 2022 ;
      53 6E D0 0772 2023 MOVL (SP),R3 ; Put TWP address in R3
      18 AE D0 0775 2024 MOVL 4+SAVED_R5(SP),R5 ; Restore UCB address
      F884' 30 0779 2025 BSBW EX$ALTQUEPKT ; Queue the request to the terminal
      50 8E D0 077C 2026 POPL R0 ; Remove TWP address from the stack
      6E 01 3C 077F 2027 MOVZWL #SS$ NORMAL,SAVED_R0(SP) ; Assume success
      20 A0 D5 0782 2028 TSTL TTY$WB_END(R0) ; Check for rejection
      05 13 0785 2029 BEQL 69$ ; Branch if request rejected
      08 14 0787 2030 BGTR 80$ ; Branch if remote terminal accepted
      3F BA 0789 2031 POPR #^M<R0,R1,R2,R3,R4,R5> ; Restore the registers
      0084 8F 05 078B 2032 RSB ; Return
      6E 3C 078C 2033 MOVZWL #SS$ DEVOFFLINE,- ; Set broadcast rejection status
      F86C' 30 0790 2034 SAVED_R0(SP) ;
      F3 11 0791 2035 BSBW COM$DRVDEALMEM ; Deallocate the TWP
      30 11 0794 2036 BRB 13$ ; Take common exit path
      30 11 0796 2037 ;
      30 11 0796 2038 ;
      30 11 0796 2039 ; The following code performs all of the necessary broadcast post-processing.
      30 11 0796 2040 ; This entry point is FORKed to at IPL IPL$-QUEUEAST from the terminal driver.
      30 11 0796 2041 ; The fork block is the TWP.
      30 11 0796 2042 ;
      50 55 D0 0796 2043 END_BROADCAST: ; Post-processor for broadcast requests
      F864' 31 0796 2044 MOVL R5,R0 ; Copy TWP address
      30 31 0799 2045 BRW EX$DEANONPAGED ; Deallocate the TWP and return
```

```
079C 2047 .SBTTL BROADCAST EMERGENCY MESSAGE TO CONSOLE
079C 2048 :++
079C 2049 : IOC$CONBRDCST
079C 2050 :
079C 2051 : This routine will allow emergency messages to be put on the console
079C 2052 : terminal. Some time later the broadcast will complete, and
079C 2053 : at that time all the necessary post-processing will be done.
079C 2054 :
079C 2055 : Input:
079C 2056 :
079C 2057 : R1 = Message length
079C 2058 : R2 = Message address
079C 2059 :
079C 2060 : Implicit input:
079C 2061 :
079C 2062 : IPL$_ASTDEL <= CURRENT_IPL <= UCBSB_FIPL(R5)
079C 2063 :
079C 2064 : A dedicated TWP block must immediately preced the message.
079C 2065 : The low bit of the first byte of the TWP is assumed to remain clear
079C 2066 : while it is in use.
079C 2067 :
079C 2068 : Output:
079C 2069 :
079C 2070 : None. The contents of R1 .. R5 are preserved across the call.
079C 2071 :
079C 2072 : Routine value:
079C 2073 :
079C 2074 : SSS$_NORMAL - The broadcast completed successfully.
079C 2075 :--
079C 2076 :
00000000 079C 2077 SAVED_R0 = 0
00000004 079C 2078 SAVED_R1 = 4
00000008 079C 2079 SAVED_R2 = 8
0000000C 079C 2080 SAVED_R3 = 12
00000010 079C 2081 SAVED_R4 = 16
00000014 079C 2082 SAVED_R5 = 20
079C 2083 :
079C 2084 IOC$CONBRDCST::
079C 2085 PUSH R0,R1,R2,R3,R4,R5 : Broadcast to a terminal
079C 2086 MOVAB OPASUCB0,R5 : Save R0 .. R5
079C 2087 SUBL2 #TTY$_WB_LENGTH,R2 : Get the console terminal UCB
079C 2088 : Retreat to the start of the TWP
079C 2089 :
079C 2090 : Fill in the Terminal Write Packet (TWP).
079C 2091 :
079C 2092 MOVW R1,TTY$_WB_SIZE(R2) : Set TWP size
079C 2093 MOV B #DYN$C_TWP, : Set TWP structure type
079C 2094 :
079C 2095 MOV B #IPL$_QUEUEAST, : Set the TWP fork IPL (for later use)
079C 2096 :
079C 2097 MOVL #1,TTY$_WB_FR3(R2) : Request refresh of read prompt
079C 2098 MOVAB TTY$_WB_DATA(R2), : Set address of message start
079C 2099 :
079C 2100 ADDL3 SAVED_R1(SP), : Set address of message end
079C 2101 :
079C 2102 MOVAB B^END_CONBRDCST, : Set callback address
079C 2103 :
079C 2104 TTY$_WB_RETADDR(R2)
079C 2105 :
079C 2106 :
079C 2107 :
079C 2108 :
079C 2109 :
079C 2110 :
079C 2111 :
079C 2112 :
079C 2113 :
079C 2114 :
079C 2115 :
079C 2116 :
079C 2117 :
079C 2118 :
079C 2119 :
079C 2120 :
079C 2121 :
079C 2122 :
079C 2123 :
079C 2124 :
079C 2125 :
079C 2126 :
079C 2127 :
079C 2128 :
079C 2129 :
079C 2130 :
079C 2131 :
079C 2132 :
079C 2133 :
079C 2134 :
079C 2135 :
079C 2136 :
079C 2137 :
079C 2138 :
079C 2139 :
079C 2140 :
079C 2141 :
079C 2142 :
079C 2143 :
079C 2144 :
079C 2145 :
079C 2146 :
079C 2147 :
079C 2148 :
079C 2149 :
079C 2150 :
079C 2151 :
079C 2152 :
079C 2153 :
079C 2154 :
079C 2155 :
079C 2156 :
079C 2157 :
079C 2158 :
079C 2159 :
079C 2160 :
079C 2161 :
079C 2162 :
079C 2163 :
079C 2164 :
079C 2165 :
079C 2166 :
079C 2167 :
079C 2168 :
079C 2169 :
079C 2170 :
079C 2171 :
079C 2172 :
079C 2173 :
079C 2174 :
079C 2175 :
079C 2176 :
079C 2177 :
079C 2178 :
079C 2179 :
079C 2180 :
079C 2181 :
079C 2182 :
079C 2183 :
079C 2184 :
079C 2185 :
079C 2186 :
079C 2187 :
079C 2188 :
079C 2189 :
079C 2190 :
079C 2191 :
079C 2192 :
079C 2193 :
079C 2194 :
079C 2195 :
079C 2196 :
079C 2197 :
079C 2198 :
079C 2199 :
079C 2200 :
079C 2201 :
079C 2202 :
079C 2203 :
079C 2204 :
079C 2205 :
079C 2206 :
079C 2207 :
079C 2208 :
079C 2209 :
079C 2210 :
079C 2211 :
079C 2212 :
079C 2213 :
079C 2214 :
079C 2215 :
079C 2216 :
079C 2217 :
079C 2218 :
079C 2219 :
079C 2220 :
079C 2221 :
079C 2222 :
079C 2223 :
079C 2224 :
079C 2225 :
079C 2226 :
079C 2227 :
079C 2228 :
079C 2229 :
079C 2230 :
079C 2231 :
079C 2232 :
079C 2233 :
079C 2234 :
079C 2235 :
079C 2236 :
079C 2237 :
079C 2238 :
079C 2239 :
079C 2240 :
079C 2241 :
079C 2242 :
079C 2243 :
079C 2244 :
079C 2245 :
079C 2246 :
079C 2247 :
079C 2248 :
079C 2249 :
079C 2250 :
079C 2251 :
079C 2252 :
079C 2253 :
079C 2254 :
079C 2255 :
079C 2256 :
079C 2257 :
079C 2258 :
079C 2259 :
079C 2260 :
079C 2261 :
079C 2262 :
079C 2263 :
079C 2264 :
079C 2265 :
079C 2266 :
079C 2267 :
079C 2268 :
079C 2269 :
079C 2270 :
079C 2271 :
079C 2272 :
079C 2273 :
079C 2274 :
079C 2275 :
079C 2276 :
079C 2277 :
079C 2278 :
079C 2279 :
079C 2280 :
079C 2281 :
079C 2282 :
079C 2283 :
079C 2284 :
079C 2285 :
079C 2286 :
079C 2287 :
079C 2288 :
079C 2289 :
079C 2290 :
079C 2291 :
079C 2292 :
079C 2293 :
079C 2294 :
079C 2295 :
079C 2296 :
079C 2297 :
079C 2298 :
079C 2299 :
079C 2300 :
079C 2301 :
079C 2302 :
079C 2303 :
079C 2304 :
079C 2305 :
079C 2306 :
079C 2307 :
079C 2308 :
079C 2309 :
079C 2310 :
079C 2311 :
079C 2312 :
079C 2313 :
079C 2314 :
079C 2315 :
079C 2316 :
079C 2317 :
079C 2318 :
079C 2319 :
079C 2320 :
079C 2321 :
079C 2322 :
079C 2323 :
079C 2324 :
079C 2325 :
079C 2326 :
079C 2327 :
079C 2328 :
079C 2329 :
079C 2330 :
079C 2331 :
079C 2332 :
079C 2333 :
079C 2334 :
079C 2335 :
079C 2336 :
079C 2337 :
079C 2338 :
079C 2339 :
079C 2340 :
079C 2341 :
079C 2342 :
079C 2343 :
079C 2344 :
079C 2345 :
079C 2346 :
079C 2347 :
079C 2348 :
079C 2349 :
079C 2350 :
079C 2351 :
079C 2352 :
079C 2353 :
079C 2354 :
079C 2355 :
079C 2356 :
079C 2357 :
079C 2358 :
079C 2359 :
079C 2360 :
079C 2361 :
079C 2362 :
079C 2363 :
079C 2364 :
079C 2365 :
079C 2366 :
079C 2367 :
079C 2368 :
079C 2369 :
079C 2370 :
079C 2371 :
079C 2372 :
079C 2373 :
079C 2374 :
079C 2375 :
079C 2376 :
079C 2377 :
079C 2378 :
079C 2379 :
079C 2380 :
079C 2381 :
079C 2382 :
079C 2383 :
079C 2384 :
079C 2385 :
079C 2386 :
079C 2387 :
079C 2388 :
079C 2389 :
079C 2390 :
079C 2391 :
079C 2392 :
079C 2393 :
079C 2394 :
079C 2395 :
079C 2396 :
079C 2397 :
079C 2398 :
079C 2399 :
079C 2400 :
079C 2401 :
079C 2402 :
079C 2403 :
079C 2404 :
079C 2405 :
079C 2406 :
079C 2407 :
079C 2408 :
079C 2409 :
079C 2410 :
079C 2411 :
079C 2412 :
079C 2413 :
079C 2414 :
079C 2415 :
079C 2416 :
079C 2417 :
079C 2418 :
079C 2419 :
079C 2420 :
079C 2421 :
079C 2422 :
079C 2423 :
079C 2424 :
079C 2425 :
079C 2426 :
079C 2427 :
079C 2428 :
079C 2429 :
079C 2430 :
079C 2431 :
079C 2432 :
079C 2433 :
079C 2434 :
079C 2435 :
079C 2436 :
079C 2437 :
079C 2438 :
079C 2439 :
079C 2440 :
079C 2441 :
079C 2442 :
079C 2443 :
079C 2444 :
079C 2445 :
079C 2446 :
079C 2447 :
079C 2448 :
079C 2449 :
079C 2450 :
079C 2451 :
079C 2452 :
079C 2453 :
079C 2454 :
079C 2455 :
079C 2456 :
079C 2457 :
079C 2458 :
079C 2459 :
079C 2460 :
079C 2461 :
079C 2462 :
079C 2463 :
079C 2464 :
079C 2465 :
079C 2466 :
079C 2467 :
079C 2468 :
079C 2469 :
079C 2470 :
079C 2471 :
079C 2472 :
079C 2473 :
079C 2474 :
079C 2475 :
079C 2476 :
079C 2477 :
079C 2478 :
079C 2479 :
079C 2480 :
079C 2481 :
079C 2482 :
079C 2483 :
079C 2484 :
079C 2485 :
079C 2486 :
079C 2487 :
079C 2488 :
079C 2489 :
079C 2490 :
079C 2491 :
079C 2492 :
079C 2493 :
079C 2494 :
079C 2495 :
079C 2496 :
079C 2497 :
079C 2498 :
079C 2499 :
079C 2500 :
079C 2501 :
079C 2502 :
079C 2503 :
079C 2504 :
079C 2505 :
079C 2506 :
079C 2507 :
079C 2508 :
079C 2509 :
079C 2510 :
079C 2511 :
079C 2512 :
079C 2513 :
079C 2514 :
079C 2515 :
079C 2516 :
079C 2517 :
079C 2518 :
079C 2519 :
079C 2520 :
079C 2521 :
079C 2522 :
079C 2523 :
079C 2524 :
079C 2525 :
079C 2526 :
079C 2527 :
079C 2528 :
079C 2529 :
079C 2530 :
079C 2531 :
079C 2532 :
079C 2533 :
079C 2534 :
079C 2535 :
079C 2536 :
079C 2537 :
079C 2538 :
079C 2539 :
079C 2540 :
079C 2541 :
079C 2542 :
079C 2543 :
079C 2544 :
079C 2545 :
079C 2546 :
079C 2547 :
079C 2548 :
079C 2549 :
079C 2550 :
079C 2551 :
079C 2552 :
079C 2553 :
079C 2554 :
079C 2555 :
079C 2556 :
079C 2557 :
079C 2558 :
079C 2559 :
079C 2560 :
079C 2561 :
079C 2562 :
079C 2563 :
079C 2564 :
079C 2565 :
079C 2566 :
079C 2567 :
079C 2568 :
079C 2569 :
079C 2570 :
079C 2571 :
079C 2572 :
079C 2573 :
079C 2574 :
079C 2575 :
079C 2576 :
079C 2577 :
079C 2578 :
079C 2579 :
079C 2580 :
079C 2581 :
079C 2582 :
079C 2583 :
079C 2584 :
079C 2585 :
079C 2586 :
079C 2587 :
079C 2588 :
079C 2589 :
079C 2590 :
079C 2591 :
079C 2592 :
079C 2593 :
079C 2594 :
079C 2595 :
079C 2596 :
079C 2597 :
079C 2598 :
079C 2599 :
079C 2600 :
079C 2601 :
079C 2602 :
079C 2603 :
079C 2604 :
079C 2605 :
079C 2606 :
079C 2607 :
079C 2608 :
079C 2609 :
079C 2610 :
079C 2611 :
079C 2612 :
079C 2613 :
079C 2614 :
079C 2615 :
079C 2616 :
079C 2617 :
079C 2618 :
079C 2619 :
079C 2620 :
079C 2621 :
079C 2622 :
079C 2623 :
079C 2624 :
079C 2625 :
079C 2626 :
079C 2627 :
079C 2628 :
079C 2629 :
079C 2630 :
079C 2631 :
079C 2632 :
079C 2633 :
079C 2634 :
079C 2635 :
079C 2636 :
079C 2637 :
079C 2638 :
079C 2639 :
079C 2640 :
079C 2641 :
079C 2642 :
079C 2643 :
079C 2644 :
079C 2645 :
079C 2646 :
079C 2647 :
079C 2648 :
079C 2649 :
079C 2650 :
079C 2651 :
079C 2652 :
079C 2653 :
079C 2654 :
079C 2655 :
079C 2656 :
079C 2657 :
079C 2658 :
079C 2659 :
079C 2660 :
079C 2661 :
079C 2662 :
079C 2663 :
079C 2664 :
079C 2665 :
079C 2666 :
079C 2667 :
079C 2668 :
079C 2669 :
079C 2670 :
079C 2671 :
079C 2672 :
079C 2673 :
079C 2674 :
079C 2675 :
079C 2676 :
079C 2677 :
079C 2678 :
079C 2679 :
079C 2680 :
079C 2681 :
079C 2682 :
079C 2683 :
079C 2684 :
079C 2685 :
079C 2686 :
079C 2687 :
079C 2688 :
079C 2689 :
079C 2690 :
079C 2691 :
079C 2692 :
079C 2693 :
079C 2694 :
079C 2695 :
079C 2696 :
079C 2697 :
079C 2698 :
079C 2699 :
079C 2700 :
079C 2701 :
079C 2702 :
079C 2703 :
079C 2704 :
079C 2705 :
079C 2706 :
079C 2707 :
079C 2708 :
079C 2709 :
079C 2710 :
079C 2711 :
079C 2712 :
079C 2713 :
079C 2714 :
079C 2715 :
079C 2716 :
079C 2717 :
079C 2718 :
079C 2719 :
079C 2720 :
079C 2721 :
079C 2722 :
079C 2723 :
079C 2724 :
079C 2725 :
079C 2726 :
079C 2727 :
079C 2728 :
079C 2729 :
079C 2730 :
079C 2731 :
079C 2732 :
079C 2733 :
079C 2734 :
079C 2735 :
079C 2736 :
079C 2737 :
079C 2738 :
079C 2739 :
079C 2740 :
079C 2741 :
079C 2742 :
079C 2743 :
079C 2744 :
079C 2745 :
079C 2746 :
079C 2747 :
079C 2748 :
079C 2749 :
079C 2750 :
079C 2751 :
079C 2752 :
079C 2753 :
079C 2754 :
079C 2755 :
079C 2756 :
079C 2757 :
079C 2758 :
079C 2759 :
079C 2760 :
079C 2761 :
079C 2762 :
079C 2763 :
079C 2764 :
079C 2765 :
079C 2766 :
079C 2767 :
079C 2768 :
079C 2769 :
079C 2770 :
079C 2771 :
079C 2772 :
079C 2773 :
079C 2774 :
079C 2775 :
079C 2776 :
079C 2777 :
079C 2778 :
079C 2779 :
079C 2780 :
079C 2781 :
079C 2782 :
079C 2783 :
079C 2784 :
079C 2785 :
079C 2786 :
079C 2787 :
079C 2788 :
079C 2789 :
079C 2790 :
079C 2791 :
079C 2792 :
079C 2793 :
079C 2794 :
079C 2795 :
079C 2796 :
079C 2797 :
079C 2798 :
079C 2799 :
079C 2800 :
079C 2801 :
079C 2802 :
079C 2803 :
079C 2804 :
079C 2805 :
079C 2806 :
079C 2807 :
079C 2808 :
079C 2809 :
079C 2810 :
079C 2811 :
079C 2812 :
079C 2813 :
079C 2814 :
079C 2815 :
079C 2816 :
079C 2817 :
079C 2818 :
079C 2819 :
079C 2820 :
079C 2821 :
079C 2822 :
079C 2823 :
079C 2824 :
079C 2825 :
079C 2826 :
079C 2827 :
079C 2828 :
079C 2829 :
079C 2830 :
079C 2831 :
079C 2832 :
079C 2833 :
079C 2834 :
079C 2835 :
079C 2836 :
079C 2837 :
079C 2838 :
079C 2839 :
079C 2840 :
079C 2841 :
079C 2842 :
079C 2843 :
079C 2844 :
079C 2845 :
079C 2846 :
079C 2847 :
079C 2848 :
079C 2849 :
079C 2850 :
079C 2851 :
079C 2852 :
079C 2853 :
079C 2854 :
079C 2855 :
079C 2856 :
079C 2857 :
079C 2858 :
079C 2859 :
079C 2860 :
079C 2861 :
079C 2862 :
079C 2863 :
079C 2864 :
079C 2865 :
079C 2866 :
079C 2867 :
079C 2868 :
079C 2869 :
079C 2870 :
079C 2871 :
079C 2872 :
079C 2873 :
079C 2874 :
079C 2875 :
079C 2876 :
079C 2877 :
079C 2878 :
079C 2879 :
079C 2880 :
079C 2881 :
079C 2882 :
079C 2883 :
079C 2884 :
079C 2885 :
079C 2886 :
079C 2887 :
079C 2888 :
079C 2889 :
079C 2890 :
079C 2891 :
079C 2892 :
079C 2893 :
079C 2894 :
079C 2895 :
079C 2896 :
079C 2897 :
079C 2898 :
079C 2899 :
079C 2900 :
079C 2901 :
079C 2902 :
079C 2903 :
079C 2904 :
079C 2905 :
079C 2906 :
079C 2907 :
079C 2908 :
079C 2909 :
079C 2910 :
079C 2911 :
079C 2912 :
079C 2913 :
079C 2914 :
079C 2915 :
079C 2916 :
079C 2917 :
079C 2918 :
079C 2919 :
079C 2920 :
079C 2921 :
079C 2922 :
079C 2923 :
079C 2924 :
079C 2925 :
079C 2926 :
079C 2927 :
079C 2928 :
079C 2929 :
079C 2930 :
079C 2931 :
079C 2932 :
079C 2933 :
079C 2934 :
079C 2935 :
079C 2936 :
079C 2937 :
079C 2938 :
079C 2939 :
079C 2940 :
079C 2941 :
079C 2942 :
079C 2943 :
079C 2944 :
079C 2945 :
079C 2946 :
079C 2947 :
079C 2948 :
079C 2949 :
079C 2950 :
079C 2951 :
079C 2952 :
079C 2953 :
079C 2954 :
079C 2955 :
079C 2956 :
079C 2957 :
079C 2958 :
079C 2959 :
079C 2960 :
079C 2961 :
079C 2962 :
079C 2963 :
079C 2964 :
079C 2965 :
079C 2966 :
079C 2967 :
079C 2968 :
079C 2969 :
079C 2970 :
079C 2971 :
079C 2972 :
079C 2973 :
079C 2974 :
079C 2975 :
079C 2976 :
079C 2977 :
079C 2978 :
079C 2979 :
079C 2980 :
079C 2981 :
079C 2982 :
079C 2983 :
079C 2984 :
079C 2985 :
079C 2986 :
079C 2987 :
079C 2988 :
079C 2989 :
079C 2990 :
079C 2991 :
079C 2992 :
079C 2993 :
079C 2994 :
079C 2995 :
079C 2996 :
079C 2997 :
079C 2998 :
079C 2999 :
079C 3000 :
079C 3001 :
079C 3002 :
079C 3003 :
079C 3004 :
079C 3005 :
079C 3006 :
079C 3007 :
079C 3008 :
079C 3009 :
079C 3010 :
079C 3011 :
079C 3012 :
079C 3013 :
079C 3014 :
079C 3015 :
079C 3016 :
079C 3017 :
079C 3018 :
079C 3019 :
079C 3020 :
079C 3021 :
079C 3022 :
079C 3023 :
079C 3024 :
079C 3025 :
079C 3026 :
079C 3027 :
079C 3028 :
079C 3029 :
079C 3030 :
079C 3031 :
079C 3032 :
079C 3033 :
079C 3034 :
079C 3035 :
079C 3036 :
079C 3037 :
079C 3038 :
079C 3039 :
079C 3040 :
079C 3041 :
079C 3042 :
079C 3043 :
079C 3044 :
079C 3045 :
079C 3046 :
079C 3047 :
079C 3048 :
079C 3049 :
079C 3050 :
079C 3051 :
079C 3052 :
079C 3053 :
079C 3054 :
079C 3055 :
079C 3056 :
079C 3057 :
079C 3058 :
079C 3059 :
079C 3060 :
079C 3061 :
079C 3062 :
079C 3063 :
079C 3064 :
079C 3065 :
079C 3066 :
079C 3067 :
079C 3068 :
079C 3069 :
079C 3070 :
079C 3071 :
079C 3072 :
079C 3073 :
079C 3074 :
079C 3075 :
079C 3076 :
079C 3077 :
079C 3078 :
079C 3079 :
079C 3080 :
079C 3081 :
079C 3082 :
079C 3083 :
079C 3084 :
079C 3085 :
079C 3086 :
079C 3087 :
079C 3088 :
079C 3089 :
079C 3090 :
079C 3091 :
079C 3092 :
079C 3093 :
079C 3094 :
079C 3095 :
079C 3096 :
079C 3097 :
079C 3098 :
079C 3099 :
079C 3100 :
079C 3101 :
079C 3102 :
079C 3103 :
079C 3104 :
079C 3105 :
079C 3106 :
079C
```

```
24 A2 D4 07C9 2104 CLRL TTY$WB_IRP(R2) : Clear pointer to associated IRP
    52 DD 07CC 2105 PUSHL R2 : Save TWP address
      07CE 2106 :
      07CE 2107 : Queue the broadcast request to the terminal.
      07CE 2108 :
53 52 D0 07CE 2109 MOVL R2,R3 : Put TWP address in R3
    FB2C 30 07D1 2110 BSBW EXE$ALTQUEPKT : Queue the request to the terminal
      50 8ED0 07D4 2111 POPL R0 : Remove TWP address from the stack
6E 01 3C 07D7 2112 MOVZWL #SS$ NORMAL,SAVED_R0(SP) : Assume success
    20 A0 D5 07DA 2113 TSTL TTY$WB_END(R0) : Check for rejection
      03 13 07DD 2114 BEQL 69$ : Branch if request rejected
      3F BA 07DF 2115 13$: POPR #^M<R0,R1,R2,R3,R4,R5> : Restore the registers
      05 07E1 2116 14$: RSB : Return
0084 8F 3C 07E2 2117 69$: MOVZWL #SS$ DEVOFFLINE,- : Set broadcast rejection status
      6E 07E6 2118 SAVED_R0(SP)
60 01 CE 07E7 2119 80$: MNEGL #1,(R0) : Mark the TWP free
      F3 11 07EA 2120 BRB 13$ : Take common exit path
      07EC 2121 :
      07EC 2122 :
      07EC 2123 : The following code performs all of the necessary broadcast post-processing.
      07EC 2124 : This entry point is FORKed to at IPL IPL$_QUEUEAST from the terminal driver.
      07EC 2125 : The fork block is the TWP.
      07EC 2126 :
      07EC 2127 :
65 01 CE 07EC 2128 END_CONBRDCST: : Post-processor for broadcast requests
      05 07EF 2129 MNEGL #1,(R5) : Mark the TWP free
      RSB
```



```
07F0 2131 .SBTTL SCAN THE I/O DATA BASE
07F0 2132
07F0 2133 :+ IOC$SCAN_IODB - Scan the I/O data base and return next block.
07F0 2134 :
07F0 2135 This routine is called to scan the device lists in the IO data base and
07F0 2136 return a pointer to the next block in the list. Context is kept in R11
07F0 2137 and by using back pointers.
07F0 2138
07F0 2139 Inputs:
07F0 2140
07F0 2141 The I/O data base is locked for read access. This means that the caller
07F0 2142 owns the I/O data base mutex and/or is at IPL SYNCH or higher.
07F0 2143
07F0 2144 R11 = 0 implies first call
07F0 2145 R11 <> 0 indicates that R11 is pointer to current DDB
07F0 2146 R10 = 0 implies end of UCB chain
07F0 2147 R10 <> 0 indicates that R10 is pointer to current UCB
07F0 2148
07F0 2149 Outputs:
07F0 2150
07F0 2151 R0 = Success status.
07F0 2152 R10 = Pointer to UCB
07F0 2153 R11 = Pointer to DDB
07F0 2154
07F0 2155 All other registers preserved.
07F0 2156
07F0 2157 :-
07F0 2158
07F0 2159 IOC$SCAN_IODB::
07F0 2160
07F0 2161 50 01 D0 07F0 2161 MOVL #1,R0 ; Success
07F0 2162 5B D5 07F3 2162 TSTL R11 ; Initial condition?
07F0 2163 2C 13 07F5 2163 BEQL 50$ ; Yes
07F0 2164 5A D5 07F7 2164 TSTL R10 ; End of chain?
07F0 2165 07 13 07F9 2165 BEQL 10$ ; Yes
07F0 2166 5A 30 AA D0 07FB 2166 MOVL UCB$LINK(R10),R10 ; Get next UCB
07F0 2167 01 13 07FF 2167 BEQL 10$ ; None
07F0 2168 05 0801 2168 RSB
07F0 2169 0802 2169
07F0 2170 6B D5 0802 2170 10$: TSTL DDB$LINK(R11) ; At end of DDB chain?
07F0 2171 0A 13 0804 2171 BEQL 30$ ; Yes
07F0 2172 5B 6B D0 0806 2172 MOVL DDB$LINK(R11),R11 ; No, get next one
07F0 2173 5A 04 AB D0 0809 2173 20$: MOVL DDB$UCB(R11),R10 ; Pick up first UCB
07F0 2174 F3 13 080D 2174 BEQL 10$ ; None, get next DDB
07F0 2175 05 080F 2175 RSB
07F0 2176 0810 2176
07F0 2177 5B 34 AB D0 0810 2177 30$: MOVL DDB$SB(R11),R11 ; Get back to parent system block
07F0 2178 5B 6B D0 0814 2178 40$: MOVL SB$FLINK(R11),R11 ; Get next system block
07F0 2179 00000000'8F 5B D1 0817 2179 CMPL R11,#SCS$GQ_CONFIG ; End of chain?
07F0 2180 0A 12 081E 2180 BNEQ 60$ ; No
07F0 2181 50 D7 0820 2181 DECL R0
07F0 2182 05 0822 2182 RSB
07F0 2183 0823 2183
07F0 2184 5B 00000000'9F D0 0823 2184 50$: MOVL @#SCS$GQ_CONFIG,R11 ; Pick up first system block
07F0 2185 54 AB D5 082A 2185 60$: TSTL SB$DDB(R11) ; Is there a DDB chain?
07F0 2186 E5 13 082D 2186 BEQL 40$ ; No, go try next SB
07F0 2187 5B 54 AB D0 082F 2187 MOVL SB$DDB(R11),R11 ; Yes, get the first DDB
```

IOSUBNPAG  
V04-000

- NONPAGED I/O RELATED SUBROUTINES<sup>K</sup> 6  
SCAN THE I/O DATA BASE

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 51  
(30)

D4	11	0833	2188	BRB	208
		0835	2189		

```
0835 2191 .SBTTL SCAN THE I/O DATA BASE BOTH PRIMARY & SECONDARY PATHS
0835 2192
0835 2193 ++ IOC$SCAN_IODB_2P
0835 2194
0835 2195 This routine is called to scan the device lists in the IO data base and
0835 2196 return a pointer to the next block in the list. Context is kept in R10
0835 2197 and R11 and by using back pointers.
0835 2198
0835 2199 SCAN_IODB_2P differs from SCAN_IODB in that it will scan both the primary
0835 2200 and secondary UCB chain for each DDB. This means that if a device is
0835 2201 dual-pathed, SCAN_IODB_2P will return the address of its UCB twice, once in
0835 2202 the context of the primary controller and once in the context of the
0835 2203 secondary.
0835 2204
0835 2205 Inputs and Outputs are identical to IOC$SCAN_IODB.
0835 2206
0835 2207
0835 2208 IOC$SCAN_IODB_2P::
0835 2209
50 01 D0 0835 2210 MOVL #1,R0 ; Success
5B 05 D5 0838 2211 TSTL R11 ; Initial condition?
41 13 083A 2212 BEQL 60$ ; Yes
5A 05 D5 083C 2213 TSTL R10 ; Caller signalled end of chain?
1C 13 083E 2214 BEQL 30$ ; Yes, done with this DDB
0840 2215
0840 2216 At this point we must decide if we're following the primary or secondary
0840 2217 chain of UCBs on this DDB.
0840 2218
5B 28 AA D1 0840 2219 CMPL UCBSL_DDB(R10),R11 ; Are we traversing the primary chain?
07 12 0844 2220 BNEQ 10$ ; Branch if we're following secondary
5A 30 AA D0 0846 2221 MOVL UCBSL_LINK(R10),R10 ; Get next UCB on primary chain
09 13 084A 2222 BEQL 20$ ; Branch if none to try secondary chain
05 05 084C 2223 RSB ; Else return UCB address to caller
084D 2224
084D 2225 Get next UCB on secondary chain.
084D 2226
5A 00A4 CA D0 084D 2227 10$: MOVL UCBSL_DP_LINK(R10),R10 ; Get next UCB on secondary chain
08 13 0852 2228 BEQL 30$ ; Branch if none left
05 05 0854 2229 RSB ; Else return UCB address to caller
0855 2230
0855 2231 No UCBs left on primary chain; traverse secondary chain if present.
0855 2232
5A 40 AB D0 0855 2233 20$: MOVL DDBSL_DP_UCB(R11),R10 ; Get first UCB on secondary chain
01 13 0859 2234 BEQL 30$ ; Branch if none to try next DDB
05 05 085B 2235 RSB ; Else return UCB address to caller
085C 2236
085C 2237 Step to next DDB.
085C 2238
085C 2239 30$: TSTL DDBSL_LINK(R11) ; At end of DDB chain?
6B D5 085C 2240 BEQL 40$ ; Yes, try next system block
0A 13 085E 2241
5B 6B D0 0860 2241 MOVL DDBSL_LINK(R11),R11 ; No, get next one
04 AB D0 0863 2242 35$: MOVL DDBSL_UCB(R11),R10 ; Pick up first UCB on primary chain
EC 13 0867 2243 BEQL 20$ ; None, try for UCB on secondary chain
05 05 0869 2244 RSB ; Else return UCB address to caller
086A 2245
086A 2246
086A 2247 Step to next system block.
```

```

      SB      34 AB      D0 086A 2248 :
00000000'8F      5B      6B      D0 086A 2249 40$: MOVL DDB$$_SB(R11),R11      : Get back to parent system block
      5B      5B      0A      D1 086E 2250 50$: MOVL SB$_FLINK(R11),R11      : Get next system block
      5B      5B      0A      D1 0871 2251      : CMPL R11,#SCS$GQ_CONFIG      : End of chain?
      5B      5B      0A      D1 0878 2252      : BNEQ 70$      : No
      5B      5B      0A      D1 087A 2253      : DECL R0      : Signal end of IO scan
      5B      5B      0A      D1 087C 2254      : RSB
      5B      5B      0A      D1 087D 2255
      5B      5B      0A      D1 087D 2256 60$: MOVL @#SCS$GQ_CONFIG,R11      : Pick up first system block
      5B      5B      0A      D1 0880 2257 70$: TSTL SB$_DDB(R11)      : Is there a DDB chain?
      5B      5B      0A      D1 0887 2258      : BEQL 50$      : No, go try next SB
      5B      5B      0A      D1 0889 2259      : MOVL SB$_DDB(R11),R11      : Yes, get the first DDB
      5B      5B      0A      D1 088D 2260      : BRB 35$      : Try for UCB on primary chain
```



```
088F 2262 .SBTTL IOC$CTRLINIT - Call driver controller init. routine
088F 2263 :++
088F 2264 : FUNCTIONAL DESCRIPTION:
088F 2265 :
088F 2266 : For UNIBUS devices, the device CSR is tested for existence. If this
088F 2267 : test fails, a no routine call occurs and failure status is returned in
088F 2268 : R0. Input values for a device driver's controller initialization
088F 2269 : routine are loaded into the proper registers, the routine starting
088F 2270 : address is located, and if a routine exists, it is called.
088F 2271 :
088F 2272 : INPUTS:
088F 2273 : R1 CSR address to use if IDB contains zero
088F 2274 : R8 CRB address (primary)
088F 2275 : R11 DDB address
088F 2276 :
088F 2277 : OUTPUTS:
088F 2278 : R0 Status (success, or failure ==> UNIBUS CSR non-existent)
088F 2279 : R1-R6 Destroyed
088F 2280 :--
088F 2281 :
088F 2282 :++
088F 2283 : Controller initialization routine parameters:
088F 2284 :
088F 2285 : INPUTS:
088F 2286 : R4 CSR address (for UNIBUS and MASSBUS devices)
088F 2287 : SCSSYSTEMID address (for class drivers during SYSGEN driver
088F 2288 : loading)
088F 2289 : zero for all others, including class drivers during power
088F 2290 : failure recovery
088F 2291 : R5 IDB address (or zero if none exists)
088F 2292 : R6 DDB address
088F 2293 : R8 CRB address
088F 2294 :
088F 2295 : OUTPUTS:
088F 2296 : Must preserve all registers except R0 through R4.
088F 2297 :--
088F 2298 :
088F 2299 :
088F 2300 :
088F 2301 IOC$CTRLINIT::
088F 2302 :
088F 2303 : MOVL CRB$SL_INTD+VEC$SL_IDB(R8), R5 : Get IDB address.
0893 2304 : BGEQ 10$ : Branch if none.
0895 2305 : MOVL IDB$SL_CSR(R5), R4 : Get CSR address.
0898 2306 : BLSS 20$ : Branch if really a CSR.
089A 2307 10$: MOVL R1, R4 : Else, use supplied value,
089D 2308 : BRB 40$ : and skip CSR testing.
089F 2309 :
089F 2310 20$: MOVL IDB$SL_ADP(R5), R6 : Get ADP address.
08A3 2311 : BGEQ 40$ : If none, skip CSR test.
08A5 2312 : CMPW #AT$_UBA, ADP$W_ADPTYPE(R6) : Is this a UBA?
08A9 2313 : BNEQ 40$ : If not a UBA, skip CSR test.
08AB 2314 : MOVL ADP$SL_CSR(R6), R6 : Get adapter config reg addr.
08AE 2315 : MOVL R4, R0 : Setup CSR for test.
08B1 2316 : JSB G^EXESTEST_CSR : Test UNIBUS CSR.
08B7 2317 : BLBC R0, 90$ : Branch if no CSR present.
08BA 2318 :
```

55	2C	A8	D0	088F	2303	MOVL	CRB\$SL_INTD+VEC\$SL_IDB(R8), R5	:	Get IDB address.
		05	18	0893	2304	BGEQ	10\$	:	Branch if none.
	54	65	D0	0895	2305	MOVL	IDB\$SL_CSR(R5), R4	:	Get CSR address.
		05	19	0898	2306	BLSS	20\$	:	Branch if really a CSR.
	54	51	D0	089A	2307	10\$: MOVL	R1, R4	:	Else, use supplied value,
		1B	11	089D	2308	BRB	40\$	:	and skip CSR testing.
				089F	2309			:	
56	14	A5	D0	089F	2310	20\$: MOVL	IDB\$SL_ADP(R5), R6	:	Get ADP address.
		15	18	08A3	2311	BGEQ	40\$	:	If none, skip CSR test.
0E	A6	01	B1	08A5	2312	CMPW	#AT\$_UBA, ADP\$W_ADPTYPE(R6)	:	Is this a UBA?
		0F	12	08A9	2313	BNEQ	40\$	:	If not a UBA, skip CSR test.
	56	66	D0	08AB	2314	MOVL	ADP\$SL_CSR(R6), R6	:	Get adapter config reg addr.
	50	54	D0	08AE	2315	MOVL	R4, R0	:	Setup CSR for test.
00000000	'GF	16	08B1	2316	JSB	G^EXESTEST_CSR		:	Test UNIBUS CSR.
	0E	50	E9	08B7	2317	BLBC	R0, 90\$	:	Branch if no CSR present.
				08BA	2318			:	

50	30	A8	D0	08BA	2319	40\$:	MOVL	CRBSL_INTD+VECSL_INITIAL(R8), R0	; Get ctrl init rout addr.
		05	18	08BE	2320		BGEQ	80\$	; Branch if none.
56	58	D0	08C0	2321			MOVL	R11, R6	; Get DDB address.
	60	16	08C3	2322			JSB	(R0)	; Call ctrl init routine.
				08C5	2323				
50	01	D0	08C5	2324	80\$:		MOVL	#1, R0	; Set success status.
		05	08C8	2325	90\$:		RSB		; Return w/ status.

```
08C9 2327 .SBTTL IOCSUNITINIT - Call driver unit init. routine
08C9 2328
08C9 2329 ++
08C9 2330 FUNCTIONAL DESCRIPTION:
08C9 2331 Input values for a device driver's unit initialization routine are
08C9 2332 loaded into the proper registers, the routine starting address is
08C9 2333 located, and if a routine exists, it is called.
08C9 2334
08C9 2335 INPUTS:
08C9 2336 R5 UCB address
08C9 2337 R8 CRB address (primary)
08C9 2338
08C9 2339 OUTPUTS:
08C9 2340 R0-R4 Destroyed
08C9 2341
08C9 2342 NOTES:
08C9 2343 There are two unit initialization routine addresses in the I/O data
08C9 2344 base; CRBSL_INTD_VECSL_UNITINIT and DDTSL_UNITINIT. Normally, only
08C9 2345 one of these two places should contain a unit initialization routine
08C9 2346 address. However, for the console block storage device, the both
08C9 2347 locations contain an address, and the DDT contains the address which
08C9 2348 must be used.
08C9 2349
08C9 2350 In this case, the CRB is shared by the console terminal and console
08C9 2351 block storage devices. The CRB contains the address of the unit
08C9 2352 initialization routine for the console terminal, and the console
08C9 2353 terminal DDT contains no unit initialization routine address. Thus
08C9 2354 the console terminal device "fits" the "normal" case. However, the
08C9 2355 console block storage device DDT contains a unit initialization
08C9 2356 routine which differs from the console terminal unit initialization
08C9 2357 routine and whose address is stored in the DDT.
08C9 2358
08C9 2359 Since the CRB is shared and contains the wrong unit initialization
08C9 2360 routine address for the console block storage device, the DDT must be
08C9 2361 inspected first. Initialization for the console terminal will be
08C9 2362 accomplished correctly regardless of which address is checked first.
08C9 2363
08C9 2364 --
08C9 2365
08C9 2366 ++
08C9 2367 Unit initialization routine parameters:
08C9 2368
08C9 2369 INPUTS:
08C9 2370 R3 CSR address (primary)
08C9 2371 R4 CSR address (secondary, same as primary if no secondary exists)
08C9 2372 R5 UCB address
08C9 2373
08C9 2374 OUTPUTS:
08C9 2375 Must preserve all registers except R0 through R4.
08C9 2376
08C9 2377 --
08C9 2378
08C9 2379
08C9 2380
08C9 2381 IOCSUNITINIT::
08C9 2382
08C9 2383 50 0088 C5 D0 08C9 2383 MOVL UCB$$_DDT(R5), R0 ; Get DDT address.
```

```

50 18 A0 D0 08CE 2384      MOVL DDT$UNITINIT(R0), R0      ; Get DDT unit init rout addr.
000005E4'8F 50 D1 08D2 2385      CMPL R0, #IOCS$RETURN      ; Null unit init routine?
06 12 08D9 2386      BNEQ 10$      ; Branch if real unit init rout.
50 3C A8 D0 08DB 2387      MOVL CRB$INTD+VEC$L_UNITINIT(R8), R0 ; Get CRB unit init rout addr.
1A 18 08DF 2388      BGEQ 90$      ; Branch if no unit init rout.
08E1 2389
54 D4 08E1 2390 10$: CLRL R4      ; Assume no IDB exists.
53 2C A8 D0 08E3 2391      MOVL CRB$INTD+VEC$L_IDB(R8), R3      ; Get IDB address.
10 18 08E7 2392      BGEQ 50$      ; Branch if none.
53 63 D0 08E9 2393      MOVL IDB$L_CSR(R3), R3      ; Get primary CSR.
54 53 D0 08EC 2394      MOVL R3, R4      ; Assume no sec. CRB exists.
51 20 A8 D0 08EF 2395      MOVL CRB$L_LINK(R8), R1      ; Get secondary CRB addr.
04 18 08F3 2396      BGEQ 50$      ; Branch if none.
08F5 2397      ASSUME IDB$L_CSR EQ 0
54 2C B1 D0 08F5 2398      MOVL @CRB$INTD+VEC$L_IDB(R1), R4      ; Get secondary CSR addr.
08F9 2399
60 17 08F9 2400 50$: JMP (R0)      ; Call unit init routine, and
08FB 2401      ; return to caller.
08FB 2402
05 08FB 2403 90$: RSB      ; No unit init routine to call:
08FC 2404      ; return to caller.
```



```
08FC 2406 .SBTTL Parse Device Name String
08FC 2407
08FC 2408
08FC 2409
08FC 2410 IOC$PARSDEVNAM - parse device name string
08FC 2411
08FC 2412 This routine parses a device name string, checking syntax and
08FC 2413 extracting node name, allocation class number, and unit number.
08FC 2414 If device type format is specified, it is converted into the internal
08FC 2415 compressed format.
08FC 2416
08FC 2417 INPUTS:
08FC 2418
08FC 2419 R8 = size of name string
08FC 2420 R9 = address of name string
08FC 2421 R10 = flags
08FC 2422
08FC 2423 OUTPUTS:
08FC 2424
08FC 2425 R0 = SS$NORMAL - valid name string
08FC 2426 R1 = SS$IVDEVNAM - invalid device name string
08FC 2427 R2 = unit number
08FC 2428 R3 = length of SCS node name at head of name string
08FC 2429 or allocation class number
08FC 2430 or device type code
08FC 2431 R8 = size of name string
08FC 2432 R9 = address of name string
08FC 2433 R10 = flags
08FC 2434 R4 - R7, R11 preserved
08FC 2435
08FC 2436
08FC 2437
08FC 2438 .ENABLE LSB
08FC 2439
08FC 2440 IOC$PARSDEVNAM::
08FC 2441 PUSH R4,R5,R6
08FC 2442 TSTL R8
08FC 2443 BEQL 30$
08FC 2444 MOVQ R8,R4
08FC 2445 SUBL3 #1,R9,R6
08FC 2446
08FC 2447
08FC 2448 LOCC #^A'$',R8,(R9)
08FC 2449 BEQL 10$
08FC 2450 MOVL R1,R6
08FC 2451 10$: CLRQ R2
08FC 2452 20$: MOVZBL (R5),R0
08FC 2453 BBC #6,R0,40$
08FC 2454 BICB #^X20,R0
08FC 2455 CMPB R0,#^A'Z'
08FC 2456 BGTRU 150$
08FC 2457 CMPB R0,#^A'A'
08FC 2458 BGEQU 70$
08FC 2459 30$: BRB 150$
08FC 2460
08FC 2461 Non alphabetic - may be numeric or '$'
08FC 2462
```

0070 8F BB 08FC 2441 PUSH R4,R5,R6 : save working registers  
58 D5 0900 2442 TSTL R8 : check name string length  
28 13 0902 2443 BEQL 30\$ : branch if null - error  
54 58 7D 0904 2444 MOVQ R8,R4 : copy name string descriptor  
56 59 01 C3 0907 2445 SUBL3 #1,R9,R6 : default is no node no allocation  
090B 2446 : class, set pointer before beginning  
090B 2447 : of the string  
69 58 24 3A 090B 2448 LOCC #^A'\$',R8,(R9) : scan name for a '\$'  
03 13 090F 2449 BEQL 10\$ : failed to find one - no nodename  
56 51 D0 0911 2450 MOVL R1,R6 : found it, save pointer  
52 7C 0914 2451 10\$: CLRQ R2 : init unit number and node name  
50 65 9A 0916 2452 20\$: MOVZBL (R5),R0 : get next character  
11 50 06 E1 0919 2453 BBC #6,R0,40\$ : br if code 0-^X3F - numeric or \$  
50 20 8A 091D 2454 BICB #^X20,R0 : collapse lower case to upper case  
SA 8F 50 91 0920 2455 CMPB R0,#^A'Z' : possible alphabetic?  
77 1A 0924 2456 BGTRU 150\$ : br if not  
41 8F 50 91 0926 2457 CMPB R0,#^A'A' : possible alphabetic?  
37 1E 092A 2458 BGEQU 70\$ : branch if OK - store it  
6F 11 092C 2459 30\$: BRB 150\$ : no - error  
092E 2460  
092E 2461  
092E 2462

```
56 55 D1 092E 2463 40$: CMPL R5,R6 ; hit the '$' yet?
    OE 13 0931 2464 ; BEQL 50$ ; yes, deal with it
    34 1A 0933 2465 ; BGTRU 80$ ; past it, digits are unit number
39 50 91 0935 2466 ; CMPB R0,#'A'9' ; legal?
    63 1A 0938 2467 ; BGTRU 150$ ; no, error
30 50 91 093A 2468 ; CMPB R0,#'A'0' ; legal?
    24 1E 093D 2469 ; BGEQU 70$ ; yes, accept it as alpha
    5C 11 093F 2470 ; BRB 150$ ; no, error
    0941 2471 ;
    0941 2472 ; $ in device name - either node name or allocation class.
    0941 2473 ;
53 55 59 C3 0941 2474 50$: SUBL3 R9,R5,R3 ; compute length of node name
    1C 12 0945 2475 ; BNEQ 70$ ; branch if non-null - process the $
    0947 2476 ;
    0947 2477 ; Process allocation class number.
    0947 2478 ;
    55 D6 0947 2479 60$: INCL R5 ; move over '$' to allocation
    54 D7 0949 2480 ; DECL R4 ; class digits.
    6A 10 094B 2481 ; BSBB GETNUMBER ; convert allocation class.
53 52 D0 094D 2482 ; MOVL R2,R3 ; store requested allocation class.
    4B 15 0950 2483 ; BLEQ 150$ ; leq zero is not legal.
5A 04 88 0952 2484 ; BISB #IOCSM_CLASS,R10 ; set allocation class flag
50 24 91 0955 2485 ; CMPB #'A'$',R0 ; was terminator a '$'?
    43 12 0958 2486 ; BNEQ 150$ ; if not, invalid device name.
5B 54 7D 095A 2487 ; MOVQ R4,R8 ; reset device name - unit size.
    54 D5 095D 2488 ; TSTL R4 ; check remaining string count
    B5 14 095F 2489 ; BGTR 20$ ; if characters remain, process them.
    3A 11 0961 2490 ; BRB 150$ ; else, invalid device name.
    0963 2491 ;
    B5 50 90 0963 2492 70$: MOVB R0,(R5)+ ; store potentially upcased character
    AD 54 F5 0966 2493 ; SOBGTR R4,20$ ; any more characters to scan?
    0969 2494 ;
    0969 2495 ; End of alpha scan. Make sure we actually got a non-null device name.
    0969 2496 ;
5B 54 C2 0969 2497 80$: SUBL R4,R8 ; subtract unit number from string
    2F 13 096C 2498 ; BEQL 150$ ; if eql no device name specified
    56 D6 096E 2499 ; INCL R6 ; point past $ in node name
55 56 D1 0970 2500 ; CMPL R6,R5 ; see if we've processed any more chars
    09 1F 0973 2501 ; BLSSU 90$ ; branch if yes
    25 5A E8 0975 2502 ; BLBS R10,150$ ; branch if physical - not valid
21 5A 06 E1 0978 2503 ; BBC #IOCSV_ANY,R10,150$ ; or if not generic search for any
    OD 11 097C 2504 ; BRB 100$ ; node name only - verify end of string
    097E 2505 ;
    097E 2506 ; Process unit number and make sure there's no trailing junk.
    097E 2507 ;
    52 D4 097E 2508 90$: CLML R2 ; init unit number to 0
    54 D5 0980 2509 ; TSTL R4 ; see if there's anything left
    0B 15 0982 2510 ; BLEQ 110$ ; branch if not
5A 01 88 0984 2511 ; BISB #IOCSM_PHY,R10 ; set physical flag
    2E 10 0987 2512 ; BSBB GETNUMBER ; convert unit number
    54 D6 0989 2513 ; INCL R4 ; return terminator to string count
    54 D5 098B 2514 100$: TSTL R4 ; reached end of string?
    OE 14 098D 2515 ; BGTR 150$ ; branch if not - error
37 5A 01 E0 098F 2516 110$: BBS #IOCSV_TYPE,R10,190$ ; branch if name is a device type
    50 01 D0 0993 2517 120$: MOVL #SS$ NORMAL,R0 ; successful parse
    0070 8F BA 0996 2518 130$: POPR #'M<R4,R5,R6' ; restore registers
    05 099A 2519 ; RSB ; and return
```

```
099B 2520 : Invalid device name
099B 2521 :
099B 2522 :
099B 2523 :
50 0144 8E D5 099B 2524 140$: TSTL (SP)+ ; pop GETNUMBER return address from stack
8F 3C 099D 2525 150$: MOVZWL #SS$-1VDEVNAM,R0 ; set invalid device name
F2 11 09A2 2526 BRB 150$-
09A4 2527 :
09A4 2528 : Routine to convert ASCII to integer
09A4 2529 :
09A4 2530 : Inputs:
09A4 2531 :
09A4 2532 : R2 assumed zero
09A4 2533 : R4 size of string
09A4 2534 : R5 starting address of string
09A4 2535 :
09A4 2536 : Outputs:
09A4 2537 :
09A4 2538 : R0 terminator character
09A4 2539 : R2 converted number
09A4 2540 : R4 size of string with number and terminator character removed
09A4 2541 : R5 address of first character after number terminator character
09A4 2542 :
09A4 2543 :
50 85 9A 09A4 2544 160$: MOVZBL (R5)+,R0 ; get next character.
50 30 82 09A7 2545 SUBB #A'0',R0 ; base it at decimal digits.
10 1F 09AA 2546 BLSSU 170$ ; branch if not a decimal digit.
09 50 91 09AC 2547 CMPB R0,#9 ; is it a digit?
0B 1A 09AF 2548 BGTRU 170$ ; branch if not a decimal digit.
52 0A C4 09B1 2549 MULL #10,R2 ; scale current unit number by 10
52 50 C0 09B4 2550 ADDL R0,R2 ; add new digit to accumulation.
EA 54 F4 09B7 2551 GETNUMBER:
04 11 09BA 2552 SOBGEQ R4,160$ ; count off a character
09BC 2553 BRB 180$ ; branch if no more characters
50 FF A5 9A 09BC 2554 170$: MOVZBL -1(R5),R0 ; restore terminator character.
00008000 8F 52 D1 09C0 2555 180$: CMPL R2,#32768 ; check number value
D2 1E 09C7 2556 BGEQU 140$ ; branch if not valid
05 09C9 2557 RSB ; return to caller.
09CA 2558 :
09CA 2559 :
09CA 2560 :
09CA 2561 : Parse device type name. We do this last because all the regular device
09CA 2562 : name validation is necessary anyway. Now we just have to do the
09CA 2563 : additional checks and pack the characters.
09CA 2564 :
50 53 D5 09CA 2565 190$: TSTL R3 ; check if we saw node or alloc class
CF 12 09CC 2566 BNEQ 150$ ; branch if so - not valid
50 55 59 C3 09CE 2567 SUBL3 R9,R5,R0 ; compute total length of string
50 58 C2 09D2 2568 SUBL R8,R0 ; compute length of unit number string
02 50 D1 09D5 2569 CMPL R0,#2 ; must be two digits
C3 12 09D8 2570 BNEQ 150$ ; branch if not - not valid
55 59 D0 09DA 2571 MOVL R9,R5 ; copy name address again
02 58 D1 09DD 2572 CMPL R8,#2 ; check minimum name length
BB 1F 09E0 2573 BLSSU 150$ ; too short - out
50 85 40 8F 83 09E2 2574 SUBB3 #A'A'-1,(R5)+,R0 ; get char and convert to compressed
53 05 11 50 F0 09E7 2575 INSV R0,#17,#5,R3 ; store compressed code
50 85 40 8F 83 09EC 2576 SUBB3 #A'A'-1,(R5)+,R0 ; get char and convert to compressed
```

53	05	0C	50	F0	09F1	2577	INSV	R0,#12,#5,R3	:	store compressed code
		03	58	D1	09F6	2578	CMPL	R8,#3	:	check how many chars left
			A2	1A	09F9	2579	BGTRU	150\$	:	string was longer than 5 - error
			0A	1F	09FB	2580	BLSSU	200\$	:	short - don't take 3rd alpha
53	50	85	40	8F	83	09FD	2581	SUBB3	:	#'A'A'-1,(R5)+,R0
				F0	0A02	2582	INSV	R0,#7,#5,R3	:	get char and convert to compressed
				CO	0A07	2583	ADDL	R2,R3	:	store compressed code
				01	8A	0A0A	2584	BICB	:	add in unit number
				FF83	31	0A0D	2585	BRW	:	clear physical flag
									:	done

200\$:



```
0A10 2587 .SBTTL Search I/O Database for Device
0A10 2588
0A10 2589 :+
0A10 2590
0A10 2591 IOC$SEARCHINT - internal I/O database search
0A10 2592
0A10 2593 This routine searches the I/O database for the specified device, using
0A10 2594 the specified search rules. Depending on the search, a lock may or may
0A10 2595 not be taken out on the device when it is found.
0A10 2596
0A10 2597 Note! While this routine is non-paged and therefore may be called at
0A10 2598 elevated IPL, the device locking code it calls is not. Therefore, only
0A10 2599 searches with IOC$V_ANY may be called from elevated IPL.
0A10 2600
0A10 2601 INPUTS:
0A10 2602
0A10 2603 R2 = unit number
0A10 2604 R3 = length of SCS node name at head of name string
0A10 2605 or allocation class number
0A10 2606 or device type code
0A10 2607 R8 = size of name string
0A10 2608 R9 = address of name string
0A10 2609 R10 = flags
0A10 2610 R11 = address to store lock value block
0A10 2611 I/O database mutex held, IPL 2
0A10 2612
0A10 2613 OUTPUTS:
0A10 2614
0A10 2615 R0 = $$$_NORMAL - device found
0A10 2616 = $$$_NOSUCHDEV - device not found
0A10 2617 = $$$_NODEVAVL - device exists but not available according to rules
0A10 2618 = $$$_DEVALLOC - device allocated to other user
0A10 2619 = $$$_NOPRIV - failed device protection
0A10 2620 = $$$_TEMPLATEDEV - can't allocate template device
0A10 2621 = $$$_DEVMOUNT - device already mounted
0A10 2622 = $$$_DEVOFFLINE - device marked offline
0A10 2623 R5 = UCB
0A10 2624 R6 = DDB
0A10 2625 R7 = system block
0A10 2626 R10 - R4, R8 - R11 preserved
0A10 2627
0A10 2628 Note: If failure, R5 - R7 point to the last structures looked at.
0A10 2629
0A10 2630 :-
0A10 2631
0A10 2632
0A10 2633 Stack use:
0A10 2634
0A10 2635 SAVR2 = 0
0A10 2636 SAVR3 = 4
0A10 2637 SAVR4 = 8
0A10 2638 SAVR8 = 12
0A10 2639 SAVR9 = 16
0A10 2640
0A10 2641
0A10 2642 .ENABLE LSB
0A10 2643
```

```
00000000
00000004
00000008
0000000C
00000010
```

```
031C 8F BB OA10 2644 IOCSSEARCHINT::
OA10 2645 PUSHF #M<R2,R3,R4,R8,R9> ; save registers
OA14 2646
OA14 2647 : Search the system blocks for a suitable node. If we are doing a search
OA14 2648 by allocation class, generic device type, or no node name is given,
OA14 2649 all system blocks qualify.
OA14 2650
57 00000000'EF DE OA14 2651 MOVAL SCS$GQ_CONFIG,R7 ; get head of SCS SB list
50 67 DO OA1B 2652 10$: MOVL SB$LINK(R7),R0 ; get next system block
00000000'8F 50 D1 OA1E 2653 MOVL R0,#SCS$GQ_CONFIG ; are we back at list head?
78 13 OA25 2654 BEQL 50$ ; branch if yes - all done
OA27 2655
57 50 DO OA27 2656 MOVL R0,R7
56 54 A7 DE OA2A 2657 MOVAL SB$DDB-DDB$LINK(R7),R6 ; pick up DDB listhead
55 56 DO OA2E 2658 MOVL R6,R5 ; make sure UCB is non-zero
OA31 2659 ; if allocation class or generic dev.
5A 06 93 OA31 2660 BITB #IOCSM_CLASS!IOCSM_TYPE,R10
27 12 OA34 2661 BNEQ 30$ ; check every system block
58 0C AE 7D OA36 2662 MOVQ SAVR8(SP),R8 ; get orig dev name descriptor
53 04 AE DO OA3A 2663 MOVL SAVR3(SP),R3 ; get node name length
1D 13 OA3E 2664 BEQL 30$ ; branch if none - go ahead
44 A7 53 91 OA40 2665 CMPB R3,SB$T_NODENAME(R7) ; check node name length
D5 12 OA44 2666 BNEQ 10$ ; branch if not
69 45 A7 53 29 OA46 2667 CMPC3 R3,SB$T_NODENAME+1(R7),(R9) ; node names match?
CE 12 OA4B 2668 BNEQ 10$ ; branch if not
OA4D 2669
OA4D 2670 : Found a suitable system block. Search its DDB list.
OA4D 2671
53 04 50 01 3C OA4D 2672 20$: MOVZWL #SS$NORMAL,R0
AE 01 C1 OA50 2673 ADDL3 #1,SAVR3(SP),R3 ; include the '$'
59 53 C0 OA55 2674 ADDL R3,R9 ; skip over the nodename
58 53 C2 OA58 2675 SUBL R3,R8 ; adjust the length
52 15 OA5B 2676 BLEQ 60$ ; if no device name, just return SB
OA5D 2677
50 66 DO OA5D 2678 30$: MOVL DDB$LINK(R6),R0 ; get address of next DDB
5A 13 OA60 2679 BEQL 80$ ; if eql end of list
56 50 DO OA62 2680 MOVL R0,R6
55 D4 A6 DE OA65 2681 MOVAL <DDB$UCB-UCB$LINK>(R6),R5 ; initialize primary UCB address
5A 20 8A OA69 2682 BICB #IOCSM_2P,R10 ; new DDB - clear secondary flag
5E 5A 01 E0 OA6C 2683 BBS #IOCSV_TYPE,R10,100$ ; branch if generic type search
07 5A 02 E1 OA70 2684 BBC #IOCSV_CLASS,R10,40$ ; branch if no class to check
3C A6 04 AE D1 OA74 2685 CMPL SAVR3(SP),DDB$ALLOCLS(R6) ; else, is allo. class right?
E2 12 OA79 2686 BNEQ 30$ ; branch if not, try next DDB
15 A6 69 58 29 OA7B 2687 40$: CMPC3 R8,(R9),DDB$T_NAME+1(R6) ; check device name
DB 12 OA80 2688 BNEQ 30$ ; if no match, try next DDB
50 14 A6 9A OA82 2689 MOVZBL DDB$T_NAME(R6),R0 ; get length of name in DDB
50 58 D1 OA86 2690 CMPL R8,R0 ; check name lengths
43 13 OA89 2691 BEQL 100$ ; if they match - OK
50 50 D7 OA8B 2692 DECL R0 ; try subtracting out controller letter
50 58 D1 OA8D 2693 CMPL R8,R0 ; and see if this matches
CB 12 OA90 2694 BNEQ 30$ ; if not, keep trying
39 5A E9 OA92 2695 BLBC R10,100$ ; branch if not physical search - OK
41 8F 15 A640 91 OA95 2696 CMPC3 DDB$T_NAME+1(R6)[R0],#A'A' ; is this controller A?
31 13 OA9B 2697 BEQL 100$ ; if so, search it
BE 11 OA9D 2698 BRB 30$ ; if not, keep looking
OA9F 2699
OA9F 2700 : End of search - no suitable device has been found
```

```
50 0908 8F 3C 0A9F 2701 :  
4D 5A 04 E1 0A9F 2702 50$: MOVZWL #SS$ NOSUCHDEV,R0 : no device found  
50 0980 8F 3C 0AA4 2703 BBC #IOCSV EXISTS,R10,140$ : branch if not seen  
50 0980 8F 3C 0AA8 2704 MOVZWL #SS$ NODEVAVL,R0 : otherwise status is not available  
46 11 0AAD 2705 BRB 140$  
0AAF 2706 :  
0AAF 2707 : To here if we're just returning a system block, with no device specified.  
0AAF 2708 :  
56 66 D0 0AAF 2709 60$: MOVL (R6),R6 : get first DDB  
55 04 A6 D0 0AB2 2710 MOVL DDB$L_UCB(R6),R5 : and first UCB  
3D 11 0AB6 2711 BRB 140$ : and return  
0AB8 2712 :  
0AB8 2713 : To here when all UCB's on a DDB have been searched.  
0AB8 2714 :  
A1 5A 01 E0 0AB8 2715 70$: BBS #IOCSV_TYPE,R10,30$ : if generic type search, try next DDB  
0ABC 2716 :  
0ABC 2717 : To here when all DDB's on a system block have been searched.  
0ABC 2718 :  
5A 06 93 0ABC 2719 80$: BITB #IOCSM_CLASS!IOCSM_TYPE,R10 : if generic type or alloc class  
0A 12 0ABF 2720 BNEQ 90$ : keep searching system blocks  
5A 09 93 0AC1 2721 BITB #IOCSM_PHY!IOCSM_LOCAL,R10 : if physical or local only  
D9 12 0AC4 2722 BNEQ 50$ : we're done  
04 AE D5 0AC6 2723 TSTL SAVR3(SP) : if there was an explicit node  
D4 12 0AC9 2724 BNEQ 50$ : we're done  
FF4D 31 0ACB 2725 90$: BRW 10$ : else go try next system block  
0ACE 2726 :  
0ACE 2727 : Found a suitable DDB. Search both its UCB lists for the right UCB.  
0ACE 2728 :  
52 6E 7D 0ACE 2729 100$: MOVQ SAVR2(SP),R2 : get unit number and device type  
54 00000000'EF D0 0AD1 2730 MOVL SCH$GL_CURPCB,R4 : get PCB address  
0AD8 2731 NEXTUCB: : re-entry for next UCB  
07 5A 05 E1 0AD8 2732 110$: BBC #IOCSV_2P,R10,120$ : branch if on primary path  
55 00A4 C5 D0 0ADC 2733 MOVL UCB$L_2P_LINK(R5),R5 : link to next secondary unit.  
04 11 0AE1 2734 BRB 130$  
55 30 A5 D0 0AE3 2735 120$: MOVL UCB$L_LINK(R5),R5 : link to next primary unit.  
11 13 0AE7 2736 130$: BEQL 150$ : branch if no more units.  
28 10 0AE9 2737 BSBB IOCTESTUNIT : is this unit ok?  
07 50 E8 0AEB 2738 R0,140$ : branch if successful  
E6 5A 04 E1 0AEE 2739 BBC #IOCSV_EXISTS,R10,110$ : keep going if we haven't seen it yet  
E3 5A E9 0AF2 2740 BLBC R10,110$ : or if not physical search  
031C 8F BA 0AF5 2741 140$: POPR #M<R2,R3,R4,R8,R9> : restore registers  
05 0AF9 2742 RSB : and return  
0AFA 2743 :  
BA 5A 05 E2 0AFA 2744 150$: BBSS #IOCSV_2P,R10,70$ : branch if secondary path already searched  
55 9C A6 DE 0AFE 2745 MOVAL <DDB$L_2P_UCB - : initialize secondary UCB address.  
0B02 2746 -UCB$L_2P_LINK>(R6),R5  
D4 11 0B02 2747 BRB 110$ : go search secondary path  
0B04 2748 :  
0B04 2749 .DISABLE LSB
```

```
OB04 2751 .SBTTL Continue I/O Database Search
OB04 2752
OB04 2753 :+
OB04 2754 :
OB04 2755 IOC$SEARCHCONT - internal I/O database search
OB04 2756
OB04 2757 This routine continues a search started with a call to IOC$SEARCHINT.
OB04 2758 It uses IOC$SEARCHINT's outputs as the starting point at which to
OB04 2759 resume.
OB04 2760
OB04 2761 INPUTS:
OB04 2762
OB04 2763 R2 = unit number
OB04 2764 R3 = length of SCS node name at head of name string
OB04 2765 or allocation class number
OB04 2766 or device type code
OB04 2767 R5 = last UCB
OB04 2768 R6 = last DDB
OB04 2769 R7 = last system block
OB04 2770 R8 = size of name string
OB04 2771 R9 = address of name string
OB04 2772 R10 = flags
OB04 2773 R11 = address to store lock value block
OB04 2774 I/O database mutex held, IPL 2
OB04 2775
OB04 2776 OUTPUTS:
OB04 2777
OB04 2778 R0 = $$$_NORMAL - device found
OB04 2779 = $$$_NOSUCHDEV - device not found
OB04 2780 = $$$_NODEVAVL - device exists but not available according to rules
OB04 2781 = $$$_DEVALLOC - device allocated to other user
OB04 2782 = $$$_NOPRIV - failed device protection
OB04 2783 = $$$_TEMPLATEDEV - can't allocate template device
OB04 2784 = $$$_DEVMOUNT - device already mounted
OB04 2785 = $$$_DEVOFFLINE - device marked offline
OB04 2786 R5 = UCB
OB04 2787 R6 = DDB
OB04 2788 R7 = system block
OB04 2789 R10 - R4, R8 - R11 preserved
OB04 2790
OB04 2791 Note: If failure, R5 - R7 point to the last structures looked at.
OB04 2792
OB04 2793 :-
OB04 2794
OB04 2795 IOC$SEARCHCONT::
OB04 2796 PUSHF #M<R2,R3,R4,R8,R9> : save registers
OB04 2797 BBCC #IOC$V_ALI,R10,10$ : check if alternate UCB in use
OB04 2798 MOVL UCB$L_DP_ALTUCB(R5),R5 : link back to other to continue
OB04 2799 10$: BRB NEXTUCB : continue search
```

05 031C 8F BB  
SA 08 E5  
SS 00AB C5 D0  
C5 11



```
OB13 2801 .SBTTL Check UCB Against Search Rules
OB13 2802
OB13 2803
OB13 2804
OB13 2805 IOC$TESTUNIT - Check UCB Against Search Rules
OB13 2806
OB13 2807 INPUTS:
OB13 2808
OB13 2809 R2 = unit number
OB13 2810 R3 = device type code
OB13 2811 R4 = PCB address
OB13 2812 R5 = UCB address
OB13 2813 R10 = flags
OB13 2814 R11 = address of lock value block
OB13 2815
OB13 2816 OUTPUTS:
OB13 2817
OB13 2818 R0 = $$$_NORMAL - eligible for use according to flags
OB13 2819 = $$$_NOSUCHDEV - wrong unit number
OB13 2820 = $$$_DEVALLOC - device allocated to other user
OB13 2821 = $$$_NOPRIV - failed device protection
OB13 2822 = $$$_TEMPLATEDEV - can't allocate template device
OB13 2823 = $$$_DEVMOUNT - device already mounted
OB13 2824 = $$$_DEVOFFLINE - device marked offline
OB13 2825
OB13 2826
OB13 2827
OB13 2828 IOC$TESTUNIT::
50 0908 8F 3C OB13 2829 MOVZWL #$$$_NOSUCHDEV,R0 ; assume wrong device
   06 5A E9 OB18 2830 BLBC R10,T0$ ; branch if not physical search
54 A5 52 B1 OB1B 2831 CMPW R2,UCB$W_UNIT(R5) ; is the unit number exactly right?
   56 12 OB1F 2832 BNEQ 70$ ; branch to error if not right.
   OB21 2833
   09 5A 01 E1 OB21 2834 10$: BBC #IOC$V_TYPE,R10,20$ ; branch if not searching for dev type
   00 ED OB25 2835 CMPZV #MSCP$V_MTYPE,N,-
   16 OB27 2836 #MSCP$V_MTYPE,D1,-
53 008C C5 OB28 2837 UCB$L_MEDIA_ID(R5),R3 ; is this the requested type?
   49 12 OB2C 2838 BNEQ 70$ ; branch if not
   SA 10 B8 OB2E 2839 20$: B1SB #IOC$W_EXISTS,R10 ; note eligible device seen
OA 3C A5 03 E1 OB31 2840 #DEV$V_CDP,UCB$L_DEVCHAR2(R5),30$ ; is this served path to a local d
55 00A8 C5 D0 OB36 2841 MOVL UCB$L_DP_ALTUCB(R5),R5 ; yes, get local path UCB address.
5A 0100 8F A8 OB3B 2842 B1SW #IOC$W_ACT,R10 ; note alternate UCB in use
   03 5A 06 E1 OB40 2843 30$: BBC #IOC$V_ANY,R10,40$ ; if SEARCHALL, finish with success.
   0091 31 OB44 2844 BRW 150$
   OB47 2845
   OB47 2846 ; Check the device reference count and allocation status.
   OB47 2847
   50 006C 8F 3C OB47 2848 40$: MOVZWL #$$$_DEVMOUNT,R0 ; check if device is already mounted
55 38 A5 13 E0 OB4C 2849 BBS #DEV$V_MNT,UCB$L_DEVCHAR(R5),100$
50 0840 8F 3C OB51 2850 MOVZWL #$$$_DEVALLOC,R0
4B 64 A5 09 E0 OB56 2851 BBS #UCB$V_MOUNTING,UCB$W_STS(R5),100$ ; branch if mount in progress
   5C A5 B5 OB5B 2852 TSTW UCB$W_REF(C(R5)) ; is reference count zero?
   19 13 OB5E 2853 BEQL 80$ ; branch if reference count is zero.
   OB 5A 07 E1 OB60 2854 BBC #IOC$V_MOUNT,R10,50$ ; if mounting...
   OA 5A 0A E0 OB64 2855 BBS #IOC$V_ALLOC,R10,60$ ; if shared mount
OC 38 A5 17 E1 OB68 2856 BBC #DEV$V_ALL,UCB$L_DEVCHAR(R5),80$ ; OK if not allocated
   03 11 OB6D 2857 BRB 60$ ; otherwise check allocation
```

```

        0B6F 2858
60 A4    34 5A E9 0B6F 2859 50$: BLBC R10,100$ ; allocate: error if not phy
        2C A5 D1 0B72 2860 60$: CMPL UCB$$_PID(R5),PCB$$_PID(R4) ; does this process own the device?
        2D 12 0B77 2861 70$: BNEQ 100$ ; branch to error if not our device.
        0B79 2862 ;
        0B79 2863 ; Check all the other miscellaneous junk that can make a device not
        0B79 2864 ; available.
        0B79 2865 ;
        06 38 50 24 3C 0B79 2866 80$: MOVZWL #$$$ NOPRIV,R0 ; check if device is spooled
        A5 06 E1 0B7C 2867 BBC #DEV$V SPL,UCB$$_DEVCHAR(R5),90$ ; branch if not
        50 0084 8F 3C 0B81 2868 IFNPRIV ALL$POOL,100$,R4 ; else, process must have ALL$POOL priv.
15 38 A5 12 E1 0B87 2869 90$: MOVZWL #$$$ DEVOFFLINE,R0 ; check if device is available
10 64 A5 04 E1 0B8C 2870 BBC #DEV$V AVL,UCB$$_DEVCHAR(R5),100$
50 21DC 8F 3C 0B91 2871 BBC #UCB$V ONLINE,UCB$$_STS(R5),100$
06 64 A5 0D E0 0B96 2872 MOVZWL #$$$ TEMPLATEDEV,R0 ; check if device is a template
        F45D' 30 0B9B 2873 BBS #UCB$V TEMPLATE,UCB$$_STS(R5),100$
        OA 50 E8 0BA0 2874 BSBW EXESCHR$DACCES ; check device protection
        0B A3 2875 BLBS R0,120$ ; continue if accessible
        0B A6 2876 ;
        0B A6 2877 ; To here on any error.
        0B A6 2878 ;
        05 5A 08 E5 0BA6 2879 100$: BBCC #IOCSV_ALT,R10,110$ ; check if alternate UCB in use
55 00AB C5 D0 0BAA 2880 MOVL UCB$$_DP_ALTUCB(R5),R5 ; link back to other to continue
        05 0BAF 2881 110$: RSB ; return
        0B B0 2882 ;
        0B B0 2883 ; We've passed all the local tests. Now try to take out the appropriate
        0B B0 2884 ; lock on the device.
        0B B0 2885 ;
        51 5B D0 0B B0 2886 120$: MOVL R11,R1 ; value block address
        05 13 0B B3 2887 BEQL 130$ ; branch if none
        61 7C 0B B5 2888 CLRQ (R1) ; initialize value block
        0B A1 7C 0B B7 2889 CLRQ 8(R1)
19 3C A5 00 E1 0B B A 2890 130$: BBC #DEV$V CLU,UCB$$_DEVCHAR2(R5),150$ ; br. if not cluster visible
        50 05 D0 0B B F 2891 MOVL #LCK$K_EXMODE,R0 ; assume exclusive lock
        0C 5A 0A E0 0B C 2 2892 BBS #IOCSV_ALLOC,R10,140$ ; branch if allocation requested
        08 5A 07 E1 0B C 6 2893 BBC #IOCSV_MOUNT,R10,140$ ; branch if not mount mode
03 38 A5 17 E0 0B C A 2894 BBS #DEV$V_ALL,UCB$$_DEVCHAR(R5),140$ ; br. if allocated
        50 04 D0 0B C F 2895 MOVL #LCK$K_PWMODE,R0 ; mount, no allocation - use PW
        F42B' 30 0B D 2 2896 140$: BSBW IOC$LOCK_DEV ; and try to take device lock
        CE 50 E9 0B D 5 2897 R0,100$
        50 01 D0 0B D 8 2898 150$: MOVL #$$$_NORMAL,R0 ; indicate success
        05 0B D B 2899 RSB
```

```
OBDC 2901      .SBTTL  IOC$THREADCRB
OBDC 2902
OBDC 2903      :++
OBDC 2904      :
OBDC 2905      : FUNCTIONAL DESCRIPTION:
OBDC 2906      :
OBDC 2907      :     This routine will thread a CRB onto the duetime chain headed by
OBDC 2908      :     IOC$CRBTMOUT.
OBDC 2909      :
OBDC 2910      : CALLING SEQUENCE:
OBDC 2911      :
OBDC 2912      :     JSB      IOC$THREADCRB
OBDC 2913      :
OBDC 2914      : INPUTS:
OBDC 2915      :
OBDC 2916      :     R3 -->  CRB
OBDC 2917      :
OBDC 2918      : OUTPUTS:
OBDC 2919      :
OBDC 2920      :     NONE
OBDC 2921      :
OBDC 2922      : --
OBDC 2923
OBDC 2924      IOC$THREADCRB::
50 00000000' 50 DD OBDC 2925      PUSHL  R0      ; Save a register
60 05 13 OBDE 2926      MOVAL  G^IOC$GL_CRBTMOUT, R0 ; Pointer to list head
50 60 D0 OBE5 2927 10$: TSTL  (R0)      ; End of the line?
F7 11 OBE7 2928      BEQL  20$      ; Yes, go add new one
60 14 A3 DE OBE9 2929      MOVL  (R0), R0 ; No, get next block
50 8ED0 05 OBEC 2930      BRB    10$      ; Try, try again
OBEE 2931
60 14 A3 DE OBEE 2932 20$: MOVAL  CRB$L_TIMELINK(R3),(R0) ; Link the new block in
50 8ED0 05 OBF2 2933      POPL  R0      ; Restore register
OBF5 2934      RSB      ; Leave
OBF6 2935
OBF6 2936
OBF6 2937      .END
```

Variable	Value	Attribute	Mode
SSBASE	= 00000001		
SSDISPL	= 00000008		
SSGENSW	= 00000001		
SSHIGH	= 00000007		
SSLIMIT	= 00000006		
SSLOW	= 00000001		
SSMNSW	= 00000001		
SSMXSW	= 00000001		
ADD_DOLLAR	000006CA	R	02
ADD_NODE	000006C0	R	02
ADPSC_NUMDATAP	= 00000010		
ADPSL_CSR	= 00000000		
ADPSL_DPQBL	= 00000018		
ADPSL_DPQFL	= 00000014		
ADPSL_MRACTMDRS	= 0000005C		
ADPSL_MRQBL	= 00000034		
ADPSL_MRQFL	= 00000030		
ADPSW_ADPTYPE	= 0000000E		
ADPSW_DPBIMAP	= 00000060		
ADPSW_MRFREGARY	= 0000015E		
ADPSW_MRNREGARY	= 00000064		
ALLOC_DESCRIP	000004DF	R	02
ALLOC_NAME	0000068D	R	02
ATS_UBA	= 00000001		
BINNUM	00000000		
BOOSGL_SPTFREN	*****	X	02
BOOSGL_SPTFREL	*****	X	02
BUGS_INCONSTATE	*****	X	02
BUGS_IVBYTEALGN	*****	X	02
BUGS_UNSUPRTCPU	*****	X	02
CANSC_AMBXDGN	= 00000002		
CANSC_DASSGN	= 00000001		
CDRPSL_BCNT	= FFFFFFFD2		
CDRPSL_FPC	= 0000000C		
CDRPSL_FQFL	= 00000000		
CDRPSL_FR3	= 00000010		
CDRPSL_FR4	= 00000014		
CDRPSL_IOQFL	= FFFFFFFA0		
CDRPSL_RUCPTR	= 00000028		
CDRPSL_UBARSCE	= 0000003C		
CDRPSW_BOFF	= FFFFFFFD0		
CLUSGL_CLUB	*****	X	02
COMSDRVDEALMEM	*****	X	02
COMMON_ALOUBAMAP	0000036D	R	02
CRBSB_MASK	= 0000000E		
CRBSL_INTD	= 00000024		
CRBSL_LINK	= 00000020		
CRBSL_TIMELINK	= 00000014		
CRBSL_WQBL	= 00000004		
CRBSL_WQFL	= 00000000		
CRBSM_BSY	= 00000001		
CRBSV_BSY	= 00000000		
DCS_DISK	= 00000001		
DDBSL_2P_UCB	= 00000040		
DDBSL_ALCOCLS	= 0000003C		
DDBSL_DP_UCB	= 00000040		
DDBSL_LINK	= 00000000		

Variable	Value	Attribute	Mode
DDBSL_SB	= 000000034		
DDBSL_UCB	= 000000004		
DDBST_NAME	= 000000014		
DDTSL_CANCEL	= 00000000C		
DDTSL_REGDUMP	= 000000010		
DDTSL_START	= 000000000		
DDTSL_UNITINIT	= 000000018		
DEALLOC_DESCRIP	0000004C6	R	02
DEVSM_MBX	= 001000000		
DEVSM_TRM	= 000000004		
DEVSV_2P	= 000000004		
DEVSV_ALL	= 000000017		
DEVSV_AVL	= 000000012		
DEVSV_CDP	= 000000003		
DEVSV_CLU	= 000000000		
DEVSV_FOD	= 00000000E		
DEVSV_MNT	= 000000013		
DEVSV_NNM	= 000000009		
DEVSV_OPR	= 000000007		
DEVSV_SPL	= 000000006		
DEVSV_TRM	= 000000002		
DIR...	= 000000001		
DISKCHK	000000198	R	02
DISPLAY_NAME	0000006AA	R R	02
DO_PMS	0000001B0	R	02
DYNRC_TWP	= 000000030		
DYNRC_UCB	= 000000010		
EMBSB_DV_ERTCNT	= 000000C10		
EMBSQ_DV_IOSB	= 000000012		
EMBSW_DV_STS	= 00000001A		
END_BROADCAST	000000796	R	02
END_CONBRDCST	0000007EC	R R	02
ERLSRELEASEMB	*****	X	02
EXDVNM	0000006E3	R	02
EXESALONONPAGED	*****	X	02
EXESALTQUEPKT	*****	X	02
EXESCHKRDACCES	*****	X	02
EXESDEANONPAGED	*****	X	02
EXESGB_CPUTYPE	*****	X	02
EXESGL_ABSTIM	*****	X	02
EXESGQ_SYSTIME	*****	X	02
EXESMOONTVER	*****	X	02
EXESTEST_CSR	*****	X	02
FULL_NAME	000000688	R	02
GETNUMBER	0000009B7	R	02
IDBSL_ADP	= 000000014		
IDBSL_CSR	= 000000000		
IDBSL_OWNER	= 000000004		
IOCSA[CLOSP	00000062B	RG	02
IOCSALODATAP	000000268	R	02
IOCSALOMAPUDA	00000031B	R	02
IOCSALOUBAMAP	000000345	RG	02
IOCSALOUBAMAPN	00000033E	RG	02
IOCSALOUBAMAPSP	0000003AF	RG	02
IOCSALOUBMAPRM	000000455	RG	02
IOCSALOUBMAPRMN	00000044E	RG	02
IOCSALTREQCOM	000000118	RG	02



IOSUBNPAG  
Symbol table

- NONPAGED I/O RELATED SUBROUTINES

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 70  
(38)

IOCSBROADCAST	0000072B	RG	02
IOCSCANCELIO	00000000	RG	02
IOCSCONBRDCST	0000079C	RG	02
IOCSCREDITUCB	*****	X	02
IOCSCTRLINIT	0000088F	RG	02
IOCSCVTDEVNAM	00000652	RG	02
IOCSDALOCUBAMAP	00000573	R	02
IOCSDELETEUCB	*****	X	02
IOCSDIAGBUFILL	0000005B	RG	02
IOCSGLCRBTMOUT	*****	X	02
IOCSGLPSBL	*****	X	02
IOCSINITIATE	000001DB	RG	02
IOCSLASTCHAN	00000020	RG	02
IOCSLASTCHAN_AMBX	00000017	RG	02
IOCSLOCKDEV	*****	X	02
IOCSMNTVER	000001D2	RG	02
IOCSM_2P	= 00000020		
IOCSM_ALT	= 00000100		
IOCSM_CLASS	= 00000004		
IOCSM_EXISTS	= 00000010		
IOCSM_LOCAL	= 00000008		
IOCSM_PHY	= 00000001		
IOCSM_TYPE	= 00000002		
IOCSPARSDEVNAM	000008FC	RG	02
IOCSRELCHAN	0000008A	RG	02
IOCSRELDATAP	00000293	RG	02
IOCSRELDATAPUDA	00000288	RG	02
IOCSRELMAPREG	0000051A	RG	02
IOCSRELMAPUDA	000004FF	RG	02
IOCSRELSCHAN	00000080	RG	02
IOCSREQCOM	00000143	RG	02
IOCSREQDATAP	00000208	RG	02
IOCSREQDATAPNW	0000021A	RG	02
IOCSREQDATAPUDA	00000228	RG	02
IOCSREQMAPREG	00000309	RG	02
IOCSREQMAPUDA	000002F4	RG	02
IOCSREQPCHANH	000000E1	RG	02
IOCSREQPCHANL	000000EA	RG	02
IOCSREQSCHANH	000000CD	RG	02
IOCSREQSCHANL	000000D7	RG	02
IOCSRETURN	000005E4	RG	02
IOCSSCAN_IODB	000007F0	RG	02
IOCSSCAN_IODB_2P	00000835	RG	02
IOCSSEARCHCONT	00000B04	RG	02
IOCSSEARCHINT	00000A10	RG	02
IOCSTESTUNIT	00000B13	RG	02
IOCSTHREADCRB	00000BDC	RG	02
IOCSUNITINIT	000008C9	RG	02
IOCSV_2P	= 00000005		
IOCSV_ALLOC	= 0000000A		
IOCSV_ALT	= 00000008		
IOCSV_ANY	= 00000006		
IOCSV_CLASS	= 00000002		
IOCSV_EXISTS	= 00000004		
IOCSV_MOUNT	= 00000007		
IOCSV_TYPE	= 00000001		
IOCSWFIKPCB	000005E5	RG	02

IOCSWFIRLCH	00000607	RG	02
IPLS_ASTDEL	= 00000002		
IPLS_IOPOST	= 00000004		
IPLS_QUEUEAST	= 00000006		
IRPSL_DIAGBUF	= 0000004C		
IRPSL_IOQFL	= 00000000		
IRPSL_MEDIA	= 00000038		
IRPSL_PID	= 0000000C		
IRPSL_SVAPTE	= 0000002C		
IRPSL_UCB	= 0000001C		
IRPSV_DIAGBUF	= 00000007		
IRPSV_CHAN	= 00000028		
IRPSV_STS	= 0000002A		
LCKSK_EXMODE	= 00000005		
LCKSK_PWMODE	= 00000004		
LOCAL_NAME	000006CC	R	02
MMGSGC_SPTBASE	*****	X	02
MNTVERPNDCHK	000001B8	R	02
MSCPSV_MTYD1	= 00000016		
MSCPSV_MTYD_N	= 00000000		
NEXTUCB	00000AD8	R	02
NO_SECONDARY	000006EB	R	02
NXTIRP	00000189	R	02
OPASUCBO	*****	X	02
PCBSL_PID	= 00000060		
PCBSQ_PRIV	= 00000084		
PDTSL_ADP	= 000000E0		
PMSEND_IO	*****	X	02
PMSSGL_IOPFMPDB	*****	X	02
PMSSSTART_IO	*****	X	02
PMSSEND	0000017A	R	02
PR\$_IPL	= 00C00012		
PR\$_SID_TYP730	= 00000003		
PR\$_SID_TYP750	= 00000002		
PR\$_SID_TYP780	= 00000001		
PR\$_SID_TYP790	= 00000004		
PR\$_SID_TYP8NN	= 00000006		
PR\$_SID_TYP8SS	= 00000005		
PR\$_SID_TYPUV1	= 00000007		
PR\$_SIRR	= 00000014		
PRVSV_ALLSPOOL	= 00000004		
PUTASCIC	00000708	R	02
PUTCHAR	00000719	R	02
PUTDOLLAR	00000716	R	02
PUTNUM	000006F0	R	02
REALLOC_CD_MAPREGS	00000561	R	02
RELDATAP_COMMON	0000029F	R	02
RELEASE	00000195	R	02
RESR0	00000008		
RESR1	0000000C		
RESR2	00000010		
RESR3	00000014		
RESR4	00000018		
SAVABS...	= 0000001C		
SAVED_R0	= 00000000		
SAVED_R1	= 00000004		
SAVED_R2	= 00000008		

IOSUBNPAG  
Symbol table

- NONPAGED I/O RELATED SUBROUTINES <sup>E 8</sup>

16-SEP-1984 00:21:15 VAX/VMS Macro V04-00  
5-SEP-1984 03:43:27 [SYS.SRC]IOSUBNPAG.MAR;1

Page 71  
(38)

SAVED_R3	=	0000000C	
SAVED_R4	=	00000010	
SAVED_R5	=	00000014	
SAVR2	=	00000000	
SAVR3	=	00000004	
SAVR4	=	00000008	
SAVR8	=	0000000C	
SAVR9	=	00000010	
SB\$\$_DDB	=	00000054	
SB\$\$_FLINK	=	00000000	
SB\$\$_NODENAME	=	00000044	
SCH\$\$_GL_CURPCB	*****	X	02
SCRLEN	00000010		
SCSS\$\$_GA_LOCALSB	*****	X	02
SCSS\$\$_GQ_CONFIG	*****	X	02
SCSS\$\$_RESUMEWAITR	*****	X	02
SECONDARY_NAME	0000069A	R	02
SS\$\$_BUFFEROVF	=	00000601	
SS\$\$_DEVALLOC	=	00000840	
SS\$\$_DEVMOUNT	=	0000006C	
SS\$\$_DEVOFFLINE	=	00000084	
SS\$\$_ILLIOFUNC	=	000000F4	
SS\$\$_INSFMEM	=	00000124	
SS\$\$_IVDEVNAM	=	00000144	
SS\$\$_NODEVAVL	=	000009B0	
SS\$\$_NOPRIV	=	00000024	
SS\$\$_NORMAL	=	00000001	
SS\$\$_NOSUCHDEV	=	00000908	
SS\$\$_TEMPLATEDEV	=	000021DC	
TTY\$\$_WB_FIPL	=	0000000B	
TTY\$\$_WB_TYPE	=	0000000A	
TTY\$\$_WB_LENGTH	=	00000030	
TTY\$\$_WB_DATA	=	00000030	
TTY\$\$_WB_END	=	00000020	
TTY\$\$_WB_FR3	=	00000010	
TTY\$\$_WB_IRP	=	00000024	
TTY\$\$_WB_NEXT	=	0000001C	
TTY\$\$_WB_RETADDR	=	0000002C	
TTY\$\$_WB_SIZE	=	00000008	
UBMDS\$\$_DATAPATH	=	00000003	
UBMDS\$\$_NUMREG	=	00000002	
UBMDS\$\$_MAPREG	=	00000000	
UCB\$\$_DEVCLASS	=	00000040	
UCB\$\$_ERTCNT	=	00000080	
UCB\$\$_FIPL	=	0000000B	
UCB\$\$_TYPE	=	0000000A	
UCB\$\$_2P_LINK	=	000000A4	
UCB\$\$_CRB	=	00000024	
UCB\$\$_DDB	=	00000028	
UCB\$\$_DDT	=	00000088	
UCB\$\$_DEVCHAR	=	00000038	
UCB\$\$_DEVCHAR2	=	0000003C	
UCB\$\$_DP_ALTUCB	=	000000A8	
UCB\$\$_DP_DDB	=	000000A0	
UCB\$\$_DP_LINK	=	000000A4	
UCB\$\$_DUETIM	=	0000006C	
UCB\$\$_EMB	=	00000094	

UCB\$\$_FPC	=	0000000C
UCB\$\$_FQFL	=	00000000
UCB\$\$_FR3	=	00000010
UCB\$\$_IOQFL	=	0000004C
UCB\$\$_IRP	=	00000058
UCB\$\$_LINK	=	00000030
UCB\$\$_MEDIA_ID	=	0000008C
UCB\$\$_OPCNT	=	00000070
UCB\$\$_PID	=	0000002C
UCB\$\$_STS	=	00000064
UCB\$\$_SVAPTE	=	00000078
UCB\$\$_BSY	=	00000100
UCB\$\$_CANCEL	=	00000008
UCB\$\$_INT	=	00000002
UCB\$\$_TIM	=	00000001
UCB\$\$_TIMOUT	=	00000040
UCB\$\$_BSY	=	00000008
UCB\$\$_DELETEUCB	=	00000010
UCB\$\$_ERLOGIP	=	00000002
UCB\$\$_MNTVERIP	=	0000000E
UCB\$\$_MNTVERPND	=	00000013
UCB\$\$_MOUNTING	=	00000009
UCB\$\$_ONLINE	=	00000004
UCB\$\$_TEMPLATE	=	0000000D
UCB\$\$_BCNT	=	0000007E
UCB\$\$_BOFF	=	0000007C
UCB\$\$_REFC	=	0000005C
UCB\$\$_STS	=	00000064
UCB\$\$_UNIT	=	00000054
VEC\$\$_DATAPATH	=	00000013
VEC\$\$_NUMREG	=	00000012
VEC\$\$_ADP	=	00000014
VEC\$\$_IDB	=	00000008
VEC\$\$_INITIAL	=	0000000C
VEC\$\$_UNITINIT	=	00000018
VEC\$\$_MAPLOCK	=	00008000
VEC\$\$_DATAPATH	=	00000005
VEC\$\$_DATAPATH	=	00000000
VEC\$\$_MAPLOCK	=	0000000F
VEC\$\$_PATHLOCK	=	00000007
VEC\$\$_MAPREG	=	00000010



+-----+  
! Psect synopsis !  
+-----+

PSECT name	Allocation	PSECT No.	Attributes														
ABS	00000000 ( 0.)	00 ( 0.)	NOPIC	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE				
\$ABSS	0000001C ( 28.)	01 ( 1.)	NOPIC	USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE				
WIONONPAGED	00000BF6 ( 3062.)	02 ( 2.)	NOPIC	USR	CON	REL	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE				

+-----+  
! Performance indicators !  
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.06	00:00:01.71
Command processing	106	00:00:00.55	00:00:04.30
Pass 1	693	00:00:31.36	00:01:37.67
Symbol table sort	0	00:00:04.39	00:00:11.34
Pass 2	403	00:00:08.26	00:00:26.97
Symbol table output	1	00:00:00.25	00:00:00.66
Psect synopsis output	0	00:00:00.01	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1234	00:00:44.90	00:02:22.68

The working set limit was 2400 pages.  
182054 bytes (356 pages) of virtual memory were used to buffer the intermediate code.  
There were 150 pages of symbol table space allocated to hold 2771 non-local and 169 local symbols.  
2937 source lines were read in Pass 1, producing 24 object records in Pass 2.  
59 pages of virtual memory were used to define 55 macros.

+-----+  
! Macro library statistics !  
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	35
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	12
TOTALS (all libraries)	47

3009 GETS were required to define 47 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:IOSUBNPAG/OBJ=OBJ\$:IOSUBNPAG MSRC\$:IOSUBNPAG/UPDATE=(ENH\$:IOSUBNPAG)+EXECMLS/LIB



0376

AH-BT13A-SE  
 VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY